THE DEVELOPMENT OF AN AERIAL ROBOTICS LABORATORY HIGHLIGHTING THE FIRST EXPERIMENTAL VALIDATION OF OPTIMAL RECIPROCAL COLLISION AVOIDANCE

by

Parker James Conroy

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

The University of Utah

August 2013

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Parker James Conroy**

has been approved by the following supervisory committee members:

Jur van den Berg	, Chair	**6-13-2013** Date Approved
Jake J. Abbott	, Member	**6-13-2013** Date Approved
Mark A. Minor	, Member	**6-13-2013** Date Approved

and by **Tim Ameel** , Chair of the Department of **Mechanical Engineering**

and by Donna M. White, Interim Dean of The Graduate School.

ABSTRACT

This thesis details the development of the Algorithmic Robotics Laboratory, its experimental software environment, and a case study featuring a novel hardware validation of optimal reciprocal collision avoidance. We constructed a robotics laboratory in both software and hardware in which to perform our experiments. This lab features a netted flying volume with motion capture and two custom quadrotors. Also, two experimental software architectures are developed for actuating both ground and aerial robots within a Linux Robot Operating System environment. The first of the frameworks is based upon a single finite state machine program which managed each aspect of the experiment. Concerns about the complexity and reconfigurability of the finite state machine prompted the development of a second framework. This final framework is a multimodal structure featuring programs which focus on these specific functions: State Estimation, Robot Drivers, Experimental Controllers, Inputs, Human Robot Interaction, and a program tailored to the specifics of the algorithm tested in the experiment. These modular frameworks were used to fulfill the mission of the Algorithmic Robotics Lab, in that they were developed to validate robotics algorithms in experiments that were previously only shown in simulation.

A case study into collision avoidance was used to mark the foundation of the laboratory through the proving of an optimal reciprocal collision avoidance algorithm for the first time in hardware. In the case study, two human-controlled quadrotors were maliciously flown in colliding trajectories. Optimal reciprocal collision avoidance was demonstrated for the first time on completely independent agents with local sensing. The algorithm was shown to be robust to violations of its inherent assumptions about the dynamics of agents and the ability for those agents to sense imminent collisions. These experiments, in addition to the mathematical foundation of exponential convergence, submits that optimal reciprocal collision avoidance is a viable method for holonomic robots in both 2-D and 3-D with noisy sensing. A basis for the idea of reciprocal dance, a motion often seen in human collision avoidance, is also suggested in demonstration to be a product of uncertainty about the state of incoming agents. In the more than one hundred tests conducted in multiple environments, no midair collisions were ever produced.

With love to my friends, family, and Maxine for all of their dedication.

"Jack of all trades, master of none,
Oft times better than a master of one "

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This thesis, the culmination of my academic development, speaks not to my own skill as much as it reflects the support I have received from my advisors, friends, and family.

The research paper discussed in this thesis is the work of myself, Daman Bareiss, Matt Beall, and Jur van den Berg. The laboratory was designed and built with the help of Dustin Webb and Kyle Crandall. I would also like to thank the robotics faculty of the University of Utah and the National Science Foundation for providing me funding through the integrative graduate education and research traineeship (NSF grant 0654414).

Their are so many other people at the University who I want to thank:
Dr. Bob Roemer, Dr. Jake Abbott, Dr. Mark Minor, Dr.s Stephen and Debra Mascaro, Dr.s Stacy and Eberhard Bamberg, Dr. Will Provancher, Dr. Dan Adams, Dr. Dave Johnson, Dr. Eric Pardyjak, Mike Knutson, Tom Slowick, Richard Kirby, Andy Pamp, and Maxine Marshall.

CHAPTER 1

INTRODUCTION

1.1 Motivation

This thesis is focused on the development of an experimental robotics research architecture performed at the University of Utah. Building a laboratory capable of pursuing aerial robotics research requires a foundation of a variety of software and hardware support modules. The field of robotics is the culmination of three distinct areas of study: electrical engineering, mechanical engineering, and computer science. As such, this thesis covers topics from each of these fields pursuant to a novel case study in which we performed a hardware implementation of noncentralized Optimal Reciprocal Collision Avoidance (ORCA). These experiments are the first validation of 2-D or 3-D ORCA on independent robots.

The Algorithmic Robotics Laboratory (ARL) was initiated with the desire to create an architecture to develop unique algorithms for robot control and motion planning and then transition said simulation work to experimental hardware implementations. The work performed by Dr. Vijay Kumar at the University of Pennsylvania's GRASP lab, and the work of Dr. Raffaello D'Andrea at the Flying Machine Arena at ETH Zürich served as motivation for the design of the Algorithmic Robotics Lab and are the groups that stand out as inspiration. These labs, most likely due to their focus on dynamics work, rely heavily upon motion capture.

Important to the work performed in the ARL is the ability for the robots to function outside of the sterile laboratory environment. This involves focusing on the important factors surrounding sensing uncertainty, primarily estimating a robots state from a set of noisy sensors and implementing algorithms without motion capture. As seen in Chapter 4, after successful experiments were conducted within the motion capture arena, the robots were brought to a larger and less controlled environment so that tests could be performed at higher velocities.

Assisting the open source movement that exists in robotics communities is certainly a motivating factor for many in the laboratory. Robot Operating System (ROS) provides

a framework for developing distributed robotics across multiple languages and hardware types. This shell environment allows for hardware implementations to be quickly developed in systems that are generally isolated by data type and communication boundaries. This software bridges the microcontroller work generally performed by electrical engineers with the algorithmic work performed by computer scientists. By establishing our modular software system within ROS, we were able to share code used in the hardware implementations of ORCA across multiple experiments. The prototype discussed in Section 3.6 was designed especially to speed up all of the laboratory's research by isolating the novel algorithms vital to each experiment.

The work highlighted in this thesis focuses on a previously unvalidated optimal reciprocal collision avoidance algorithm (ORCA). This implementation demonstrated the completion of a functional laboratory. Due to the lack of a universally accepted communication method between robots, collision avoidance must be able to function when robots do not necessarily communicate. The miniaturization of electrical sensors such as inertial measurement units (IMU) has led to a boom of robot creation. As the number of autonomous vehicles increases, new and more efficient methods for collision avoidance need to be implemented so that the finite amount of airspace and ground space can be optimally utilized. As a case study, ORCA was implemented to test the qualities of the laboratory.

Such a collision avoidance framework, as developed in Section 3.6, operates optimally with a cooperating robot that takes half the responsibility for avoiding imminent collisions. ORCA expands on the idea of the relative velocity paradigm, where only locally sensed information can be used to avoid collision. This method also avoids collisions with robots that have not even detected the oncoming agent, allowing robots featuring ORCA to safely operate around a variety of other robots.

1.2 Research Objectives

1. Create the Algorithmic Robotics Lab with the necessary computers, motion capture, communication hardware, and safety equipment.

2. Build a software structure to implement novel algorithms onto a variety of robots.

3. Develop an aerial robot with which to test the lab environment.

4. Implement collision avoidance work previously only shown in simulation.

1.3 Thesis Organization

The objective of this document is to, first, discuss the design considerations that enabled the creation a viable robotics laboratory. Secondly, this thesis discusses the pinnacle of our research, where quadrotors were used for hardware validation of the ORCA algorithm.

Chapter 2 is a literature review of the research upon which this work was built. An overview of unmanned agents and aerial robots is given to establish the current state of the art in the field. Other established aerial robotics laboratories are discussed, in addition to common hardware used on aerial robots.

Chapter 3 reviews the structure of the experimental environment. Descriptions of the governing logic behind the software and hardware are given for both of the prototypes. Quadrotors are the principle robot used in the lab. These robots are discussed in detail to illustrate the controls and dynamics used in the hierarchical feedback loops that exist within even a simple experiment. Because the final prototype is used in the collision avoidance case study, a special focus is given to the final version of the software experimental environment.

Chapter 4 is the 2013 IEEE Intelligent Robots and Systems conference submission *3-D Reciprocal Collision Avoidance on Physical Quadrotor Helicopters with On-Board Sensing for Relative Positioning* verbatim. In this section, the final software architecture prototype is validated in practice.

Chapter 5 speaks to the parameters used in the ORCA experiments. The design considerations for the algorithmic parameters and the robot's specifications are discussed.

Chapter 6 contains the conclusions of the work submitted in this thesis and our considerations for the continuation of said work.

CHAPTER 2

LITERATURE REVIEW

The concepts used in Algorithmic Robotics Lab experiments feature the work of a remarkably long lineage of scientists and engineers in many fields. The following introduction to each of these subfields is provided to establish the foundation on which this work was able to be completed.

2.1 Overview of Unmanned Agents

Robots that operate outside the narrow confines of stationary assembly and manufacturing arms are often called unmanned mobile agents or unmanned systems (UMS). These robotic systems generally function as a distributed network of computers and microcontrollers which hierarchically manage the robot's functions. Hierarchical control connects high-level planning to the low-level commands. For example, for an aerial robot, the trajectory of a robot through an environment is managed at a higher level than the controls needed to keep the robot functioning stably. Often, external computing such as a wireless server is needed to process all the data generated by a mobile robot.

Unmanned systems are generally classified by their size and autonomy. Concerning autonomy metrics, the Autonomy Levels for Unmanned Systems (ALFUS) [1, 4] was developed in 2003 by the National Institute of Standards and Technology (NIST) in conjunction with United States Department of Defense. Developed for US government use in response to the growing need for the quantification of robotic capabilities, ALFUS provides a framework for the analysis of all unmanned missions undertaken by any robot, ranging from NASA ' s Curiosity to the US Armys MQ-1C Grey Eagle. The abilities of an UMS are defined according to these three dimensions of Environmental Difficulty, Mission Complexity, and Human Interface. As shown in Figure 2.1, the capabilities of a robot are defined as the volume encompassed by the three skill metrics given on the graph's axes. By definition, a robot is capable of performing any job, represented as a point in the three-dimensional space, within the robot's *skill volume*. For example, the study discussed in Chapter 4

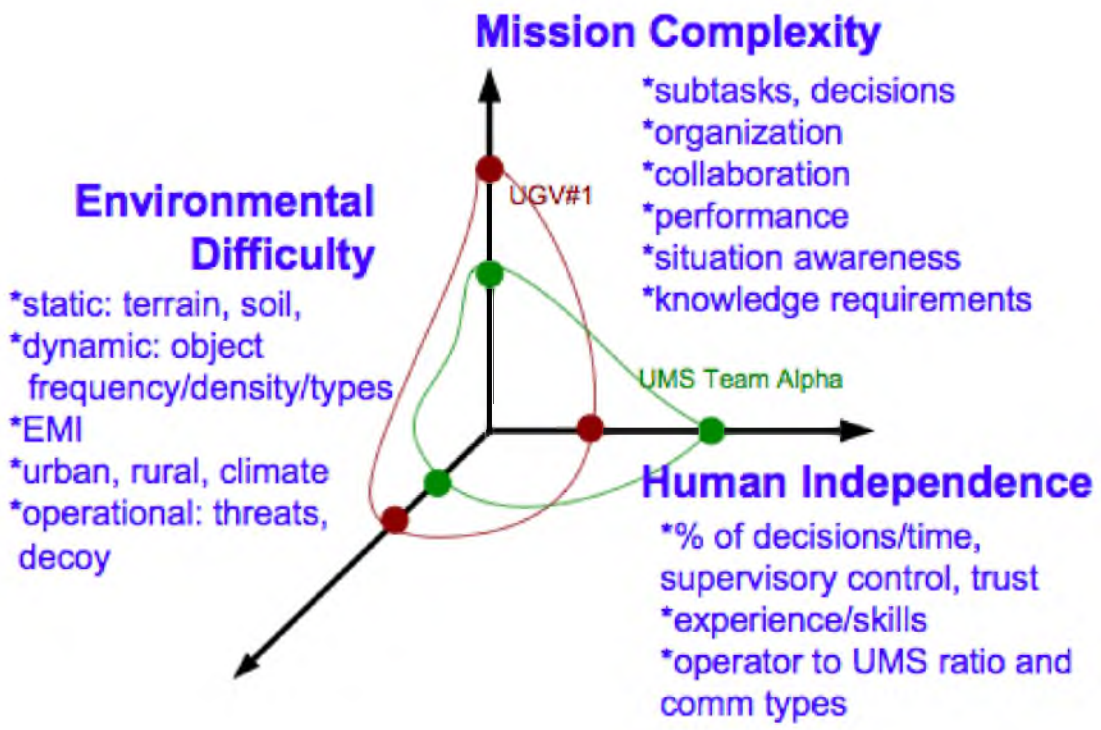


Figure 2.1. ALFUS mission space [1]. Reproduced according to [2]

features two quadrotor robots performing robotic tasks at level three, as seen in Figure 2.2. The on-board logic is designed so that a high-level desired direction is only required, as the robot independently manages the low-level collision avoidance trajectories. Remote operation of these crafts through video-assisted teleopertation is required as the robots are not able to autonomously navigate complex environments.

2.1.1 Human Robot Interaction

One of the ALFUS metrics speaks to the amount of human interaction needed for a robotic agent to be able to complete a task. Human robot interaction (HRI) is the field of research focused on analyzing the efficient use of robots by their operators [5], and developed to improve interaction between robots and their human handlers, to insure the completion of a task more effectively [6]. In an attempt to improve the usefulness of the entire laboratory setup, the experimental software (Section 3.5 and 3.6) features inputs to visualizers programmed especially to format and display information in an effective manner. Section 4.2.1 features an implementation of the ideas discussed in this section wherein a virtual environment is used to simulate the instantaneous and filtered belief of the colliding robot's state.

Human robot interaction, a subsection of human machine interaction (HMI), is crucial to roboticists because it directly impacts the value of the robots they create. No matter how capable a robot may be, a robotic system developed without an understanding of the human environment in which it operates will never reach its full potential. Gibson's theory of affordances [7] claims that all the information needed for one to act appropriately is inherent in the environment and agents in that environment. Applying this theory to robotics implies that an operator's decisions are based only on the operator's perception of the robot's abilities in the remote environment [8]. For example, robot handlers must be presented with sensor data sculpted for effective analysis by a human, in addition to control platforms that allow for simple communication between the robot-human team. An important validation of this work was performed by Stubbs [9] in an experiment using a mobile robot having varying degrees of autonomy. The results of this study focused on the inefficiencies stemming from *common ground* and situational awareness problems. "Findings suggest that as autonomy increases, users' inability to understand the reasons for the robot's actions disrupts the creation of common ground"[9]. As the HRI level increases, the required human input decreases. The control interface for the ARL laboratory was carefully developed to avoid a *common ground* discrepancy between the robot's understanding of the environment and the researcher's belief of the robot's understanding.

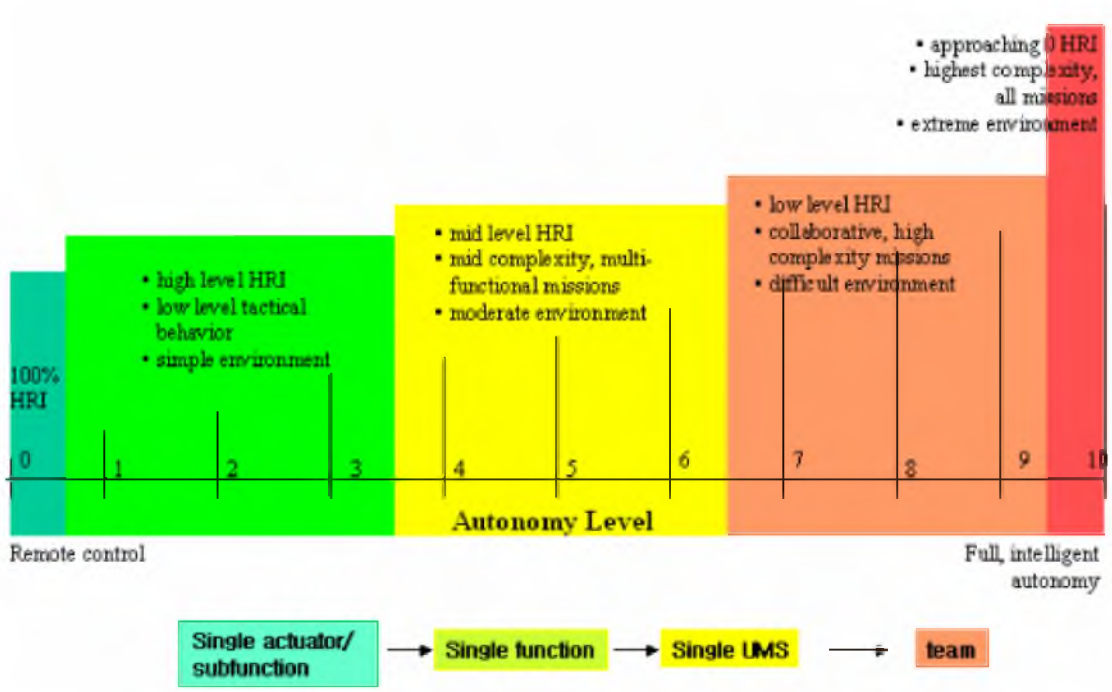


Figure 2.2. ALFUS Ability Levels [1]. Reproduced according to [2]

As explained in Section 3.6, the final experimental architecture contains specific nodes relating to HRI. Avoiding the *common ground* discrepancy involves giving researchers the ability to ascertain the state of the experiment and why certain actions are taking place. Often in an experiment, it is difficult to determine if the robot is being controlled entirely by a human, or if, for example, a collision avoidance algorithm is varying the trajectory. We believe that work is able to be completed more efficiently when researchers have a real-time understanding about what is going on algorithmically in the experiments. It is for this reason that HRI plays such an important role in the software.

2.1.2 Robotics Laboratories

Various other laboratories conduct similar work to that done at the Algorithmic Robotics Lab. Notably, the GRASP laboratory [10], directed by Dr. Kumar, features a netted environment with motion capture. Their setup consists of an array of Ascending Technologies hummingbird quadrotors controlled from computers running a MATLAB-ROS bridge.

The Flying Machine Arena at Zürich directed by Dr. D'Andrea features one of the larger aerial flying volumes at 10m x 10m x 10m [11]. The X3D robots used by the lab feature two wireless radios and are controlled at 70 Hz. In this lab, an autonomous quadrotor is used to calibrate the system, which would otherwise be very difficult to calibrate due to the laboratory's large motion capture volume.

The Robotics and Intelligent Machines Laboratory at the University of California at Berkeley features a aerial robotics group that uses a larger helicopter for outdoor research. Their experiments focus on hierarchical control management and multi-agent predator prey work [12].

Bristol Robotics Laboratory, constructed January 2013, features a flying volume (5m x 12m x 4m) focused on aerial robotic vehicle autonomy and control. Managed by Dr. Richards, formerly of MIT [13], the lab consists of a Vicon motion capture setup and uses AR.Drone robots.

Thanks to the work performed by these research laboratories, the ARL was built with a combination of each of the abilities featured. As explained in detail in Section 3.1.2, the laboratory was constructed with a netted motion capture volume featuring the ability to perform distributed robots work in ROS.

2.2 Quadrotor Mechanics

While many robots have been used in the ARL, quadrotors will be highlighted specifically because they were the robots chosen for the case study in optimal reciprocal collision

avoidance. Quadrotors are omnidirectional aerial rotorcraft composed of four fixed-pitch propellers rigidly attached in a cross formation, as seen in Figure ??. The following sections will speak to the control framework surrounding the use of quadrotors in the laboratory, and the mechanical and dynamic properties of such robots.

2.2.1 State of the Art in Aerial Robotics

An overview of aerial vehicles [14] speaks to the important distinctions between the types of miniature aerial autonomous vehicles and their development. The report concluded that two styles, helicopters (including multirotors) and blimps, have the greatest advantages concerning indoor aerial robots. Although helicopters require complex controls and are less energy efficient than fixed wing aircraft, their ability to scale down in size while maintaining maneuverability, notably vertical take off and landing (VTOL), makes helicopters well-suited for small-scale robotics research. This maneuverability is solely a function of the robot's degrees of freedom. No standards exist for qualifying the size of unmanned airframes, but there appears to be three generally accepted size ranges [15] for small aerial robots: micro Unmanned Aerial Vehicles (mUAVs) ranging from approximately two hundred grams to one kilogram, small UAVs ranging from one kilogram up to five kilograms, and finally full-size or tactical UAVs at over five kilograms. Aerial robots, due to the large power requirements for lifting the sensor payload, generally have autonomous capabilities proportional to their size. However, this does not hold for robots used in regulated environments such as motion capture volumes or robotic systems that require off-vehicle sensor processing. Currently a wide range of quadrotors exists in each of these size/ability classifications. For example, the midsize 900 gram Pennsylvania State quadrotor [16] shows semi-autonomous capabilities while keeping the flying time reasonable for research needs. Furthermore, Stanford's 1.5 kilogram quadrotor, Starmac [17, 18], has a full sensor suite that enables fully autonomous capabilities.

2.2.2 Electro-Mechanical Components

A quadrotor has no control surfaces, such as flaps or ailerons, and is only controlled through the thrust created by each of its propellers. One must realize also that a dynamic model used for any robot is only a model of how one actually operates, and may not take into account the intricacies of each of its mechanical components. Nevertheless, knowledge of the underlying principles of these components allows a fuller understanding of the robot.

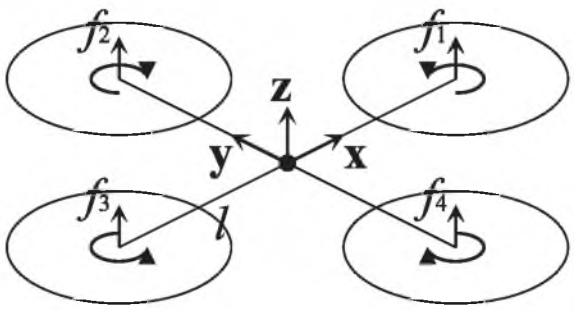


Figure 2.3. Schematic figure of a quadrotor helicopter.

2.2.2.1 Propellers

Each individual propeller of a quadrotor generates thrust based upon a series of factors specific to the propeller geometry and fluidic environment [19] [20]. The thrust of a propeller is a function of the air density ρ, the propeller constant C_T, the area encompassed by the rotating propeller A_r, and the rotational velocity ω, a propeller constant C_t combining geometric factors.

$$T = C_t \rho A_r r^2 \omega^2 \tag{2.1}$$

Through the use of a testing fixture, each of the variables in this equation can be measured. Once these factors are known, a quadratic thrust profile for the propeller can be developed.

This model represented by Equation 2.1 takes into account only the static forces generated by a propeller. As developed further in Section 2.3, a propeller moving through the air creates a variety of other aerodynamic effects pertinent to the development of the dynamic quadrotor model. Another important effect is the change in thrust as a function of air pressure and the limitations on thrust at altitude.

2.2.2.2 Brushless Direct Current Motors

Brushless direct current motors (BLDCM) are generally used in the propulsion of quadrotors. They offer many advantages over brushed motors in high-speed applications [21], the foremost being the construction consisting of no wearing components besides bearings. This allows higher motor speeds with less electrical noise. Downsides to BLDCMs include the increased complexity of the motor controller compared to the controllers used with brushed motors. The angular position of the sensorless motors used on the ARL quadrotor are determined by the back electromotive force (EMF) in nonpowered motor windings. While BLDCMs are driven via alternating current, the torque (Equation 2.2) created by a brushless motor can be defined in terms of steady state voltages [22]. Equation 2.2 can be seen as the sum of the product of the voltage and current of each winding divided by the speed of the motor. The angular acceleration is a function of the motor geometry and the load it carries.

Where:

e is the steady state winding voltage

i is the current in a motor winding

ω is the motor speed

T_e is the motor torque

T_l is the torque carried by the motor

J is the rotational inertia of the motor and load

B is the motor damping

$$T = (e_a i_a, + e_b i_b + e_c i_c)/\omega_r \tag{2.2}$$

$$\dot{\omega_r} = (T_e - T_l - B\omega_r)/J \tag{2.3}$$

Unlike brushed motors, which can be driven effectively with an H-bridge using constant voltages, BLDCMs require an alternating current [23] across the stator windings to actuate the shaft. This motor level signal is generated with a motor controller from a microcontroller level pulse width-modulated (PWM) input.

2.3 Rotorcraft Dynamics

Unlike traditional aircraft, multirotors feature no control surfaces. A quadrotor is controlled entirely from the relative thrust created by each of the four motors. The state of a quadrotor consists of six degrees of freedom, three in position and three in orientation. While the robot may move in 6 dimensions, it consists of only four actuators. This discrepancy between the input and output spaces leads to under-actuated dynamics in which states are coupled. Two of the propellers (the source of thrust labeled f_1 and f_3 in Fig. 2.3) rotate clockwise while the other two are counter-rotating propellers. Changing the speed of these motors allows the torque about the **z** axis to be controlled. More specifically, the quadrotor's state is controlled solely by the absolute and relative magnitude of the thrust from each of the four propellers. Hover is achieved when the thrust of the motors provides a force equal and opposite to that of the weight of the robot. A thrust differential across either of the two opposing rotors causes a change in attitude, rotating the quadrotor into an angle of attack relative to the gravity vector. This results in thrust vectoring, and thus an acceleration in the global frame. Generally, absolute motor thrust is increased from the base hover level to allow the quadrotor to fly without losing altitude. Rotations in the local x-y plane are controlled by differential thrusting between the pairs of similarly rotating propellers. Quadrotors are often flown in one of two configurations: plus, where one propeller flies in front of the center of the craft, and cross, where two rotors fly on the leading edge. In the ideal design, the center of the mass is located at the origin (Fig. 2.3).

Quadrotor aerodynamics are especially interesting due to fluid dynamic interactions between the leading and following edges of each propeller. Another set of interactions are due to the fluid flow between the leading and following sets of propellers. These effects come into play in the propeller configurations where the propeller's angular velocity is close

to parallel with the interacting fluid's velocity. One of these propeller effects, bird flapping, is explained in Dr. Leishman's study of helicopters [19]. A moment on the motor shaft is caused by a pressure differential between the fluid encountering the leading edge of the propeller and the fluid encountering the following edge of the propeller. Conveniently, multirotors can counteract this twisting moment with additional thrust from the opposing propeller. The second effect is a pressure drop across the front and back sets of rotors. Laminar air impacts the front rotor(s) leaving a turbulent wake for the second rotor(s). At high speeds, this moment, caused by the increase in the thrust of the front rotors relative to the back rotors, twists the UAV to a smaller angle of attack than expected. Quadrotor models for flight at high speeds must take this effect into account to fly effectively.

As a result, two additional significant effects are added to the basic dynamics to improve the accuracy of the model. Rotor drag (k_d of Eq. 2.6) occurs as a result of air moving through the propellers along the plane represented by the vectors $\mathbf{x}$ and $\mathbf{y}$ in Figure 2.3. The second effect, induced flow (k_i of Eq. 2.6), is that of air moving through the propeller along the $\mathbf{z}$ vector, which creates additional thrust when flying down, and less thrust when flying upward. Modeled in detail by Dr. Martin [24], these forces increase linearly with the velocity of the fluid.

Viscous aerodynamic effects, which increase quadratically with speed, have a minimal effect on the accuracy of the model at the speeds used in the laboratory, and hence are not covered here. An in-depth analysis of quadrotor aerodynamics is given by Dr. Tomlin's lab [18], which models the flight of quadrotors at high speeds. In striking similarity to fixed wing aircraft, quadrotors are found to experience *stall*, which is a dynamic effect concerning angle of attack at high speed. Because of the large thrusts needed for a quadrotor to travel upward quickly, the lack of thrust in the forward direction can cause a quadrotor to fall. The upward limit on these two speeds, due to the propeller and motor geometry, is increasingly sensitive to large angles of attack due to the coupled attitude control. Expanding on the effect of induced flow submitted by Dr. Martin, the paper investigates the descent of a quadrotor at speed, asserting the existence of three distinct methods of descent. Knowledge of these aerodynamic regions has important impacts on the upper limit of descent speeds for a quadrotor.

The dynamics of a quadrotor can be seen as the culmination of the effects explained above. In this, the acceleration is the sum of the motor forces minus the linear motor drag in x and y directions, and the induced flow in the z direction. The change in attitude is the result of the angular velocity, and its gyroscopic effects. The angular acceleration is

due to the torques created by the sets of motors which rotate in different directions, where two of the motors induce a clockwise rotation, and the other two induce a counterclockwise rotation. Roll and pitch are a product of the balance between these sets of propellers on a set by set basis, in the direction they exist on the quadrotor. The force given by the motors is a function of the desired force and the current motor force, and a constant that speaks to the time it takes for the propellers to change speed. The dynamic quadrotor model [25] is described in state space form as follows: $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ where the state vector $\mathbf{x} \in \mathbb{R}^{16}$ and control input vector $\mathbf{u} \in \mathbb{R}^4$ are defined by:

$$\mathbf{x} = [\mathbf{p} \quad \mathbf{v} \quad \mathbf{r} \quad \mathbf{w} \quad \mathbf{f}]^T, \qquad \mathbf{u} = \mathbf{f}^* \tag{2.4}$$

$$\dot{\mathbf{p}} = \mathbf{v} \tag{2.5}$$

$$\begin{aligned}\dot{\mathbf{v}} = &-g\mathbf{z} + \hat{R}\exp([\mathbf{r}])\big((f_1 + f_2 + f_3 + f_4)\mathbf{z} - \\ &k_d(\mathbf{v} \cdot \hat{R}\exp([\mathbf{r}])\mathbf{x})\mathbf{x} - k_d(\mathbf{v} \cdot \hat{R}\exp([\mathbf{r}])\mathbf{y})\mathbf{y} - \\ &k_i(\mathbf{v} \cdot \hat{R}\exp([\mathbf{r}])\mathbf{z})\mathbf{z}\big)/m\end{aligned} \tag{2.6}$$

$$\dot{\mathbf{r}} = \mathbf{w} + \frac{1}{2}[\mathbf{r}]\mathbf{w} + (1 - \frac{\|\mathbf{r}\|}{2\tan(\|\mathbf{r}\|/2)})\frac{[\mathbf{r}]^2}{\|\mathbf{r}\|^2}\mathbf{w} \tag{2.7}$$

$$\begin{aligned}\dot{\mathbf{w}} = &J^{-1}(l(f_2 - f_4)\mathbf{x} + l(f_3 - f_1)\mathbf{y} + \\ &k_m(f_1 - f_2 + f_3 - f_4)\mathbf{z} - [\mathbf{w}]J\mathbf{w}\big)\end{aligned} \tag{2.8}$$

$$\dot{\mathbf{f}} = k_f(\mathbf{f}^* - \mathbf{f}) \tag{2.9}$$

Where:

$\mathbf{p}$ is the position of the quadrotor in the global $\mathbb{R}^3$ coordinate frame.

$\mathbf{v}$ is the velocity of the quadrotor in the global $\mathbb{R}^3$ coordinate frame.

$\mathbf{w}$ is the angular velocity of quadrotor in local $\mathbb{R}^3$ coordinate frame.

z is the vector $[0, 0, 1]^T$.

x is the vector $[1, 0, 0]^T$.

y is the vector $[0, 1, 0]^T$.

f_i is force produced by the i'th rotor.

f is force produced by all the rotors.

f_i^* is the desired force produced by the i'th rotor.

g is the scalar representing gravity.

J is the moment of inertia matrix of the quadrotor in local $\mathbb{R}^3$ frame.

k_d is the rotor drag constant.

k_f is the rotor latency constant.

k_i is the induced flow constant.

k_m is the motor force to thrust constant.

l is the distance between the center of mass and the center of each rotor of the quadrotor.

m is the mass of quadrotor.

$\bar{R}$ is the reference rotation matrix from local to global coordinate frame.

$\mathbf{r}$ is the rotation vector of quadrotor relative to reference orientation.

The notation $[\mathbf{a}]$ for a vector $\mathbf{a} = [a_x, a_y, a_z]^T \in \mathbb{R}^3$ refers to the skew-symmetric cross-product matrix.

2.4 Robot Operating System

Robot Operating System (ROS) is an open source project maintained by Willow Garage [26] that provides a graph-like structure for distributed robotics. Originally developed in 2007 by Standford under the name Switchyard [27], ROS is a collection of interprogramming language headers that allows the sharing of data between independent programs, referred to in ROS nomenclature as nodes. We chose to use ROS because its structure allows for one to use functional programming techniques on a larger scale. Nodes can be written independently of their role in the experimental framework, and thus can operate as standalone black-boxes. These nodes can also be shared and distributed between researchers, allowing one to focus only on the novel methods.

Robot Visualizer (RVIZ) [28] is a built-in ROS 3D environment used for displaying information in ROS. RVIZ is able to natively visualize six DOF ridged agents, lines, and point clouds in addition to image streams from cameras.

2.5 Collision Avoidance

Once ORCA has been validated as a viable collision avoidance method in hardware, it exists in same framework as many other collision avoidance methods that have been previously validated and are currently used in practice. While it remains the only reciprocal collision avoidance method, that is, an algorithm in which agents share responsibility for avoiding collision, the end result is comparable to other systems that allow agents to avoid collision. In this, ORCA is related to many motion planning algorithms where agents are assumed to be obstacles.

Methods relating to potential fields [29] have been used for multi-agent collision avoidance. Priority queues, where more important robots view all less important as static

obstacles [30], use centralized methods to plan movement. Particle-based methods [31] are often used for situations where uncertainty is taken into account. Sampling-based methods have also been incorporated as summarized in [32].

2.6 Reciprocal Collision Avoidance

The following section is an introduction to the methods used in the optimal collision avoidance case study. In these experiments, the ORCA algorithm was implemented on two quadrotors to validate its functionality with dynamics not featured in previously proven simulation results. Furthermore, neither 2-D or 3-D reciprocal collision avoidance has been previously experimentally verified on independent agents with distributed on-board sensing.

Reciprocal collision avoidance (RCA) has been an actively studied area in robotics for the past few years. The problem can generally be defined in the context of multiple autonomous mobile robots navigating a common environment, where each robot employs a decentralized continuous sensing-control cycle. In each cycle, each robot must independently compute an action based on its local observations of the other robots, without mutual communication or coordination, such that it stays collision free while progressing toward a goal. The key aspect of most reciprocal collision avoidance approaches is that they specifically account for the reactive nature of the other robots, assuming each robot takes half the responsibility of avoiding pairwise collisions. Failing to do so would inherently cause undesirable oscillations in the motion of the robots [33]. Basic reciprocal collision avoidance approaches apply to robots with simple holonomic dynamics [33, 34], and more recently RCA has been extended to robots with differential-drive dynamics [35, 36], car-like dynamics [37], double-integrator dynamics [38, 39], and general linear dynamics [40].

An appealing feature of optimal reciprocal collision avoidance (ORCA) [34] is that it is based on the *relative velocity* paradigm [41]. That is, in addition to absolute local velocity data, each of the robots only need information about the *relative* position and the *relative* velocity of the other robots. In principle, this makes the approach well-suited for use in environments where absolute position information is unavailable, as the robots can acquire relative position information from on-board sensors.

The concept of reciprocal collision avoidance was first introduced in [33], which presented Reciprocal Velocity Obstacles as an extension to Velocity Obstacles (VO) [42], where agents actively attempt to avoid collisions with each other. While this approach overcame the oscillations observed with VO and guarantees collision avoidance for a pair of robots, it still exhibited oscillations in settings with more than two robots. Hybrid RVO [43, 44] was introduced to mitigate this undesired behavior, but does not offer any formal guarantees on

smoothness and collision avoidance. Optimal Reciprocal Collision Avoidance (ORCA) [34] addressed these issues, and formal proofs of smoothness and collision avoidance in settings with an arbitrary number of robots were given in [35].

The aforementioned approaches all focus on robots with idealized, holonomic dynamics, i.e. robots that can instantaneously adopt any velocity in the 2-D plane. While this assumption can be made applicable to differential-drive robots by enlarging the effective robot radius [35], it does not generalize to robots with more involved dynamics. Acceleration-velocity obstacles (AVO) [38] partly overcome this, and guarantee collision avoidance for robots with double-integrator dynamics (i.e. omni-directional acceleration control). This formulation can be applied to robots with car-like dynamics using a change of variables in the dynamics. Alonso-Mora et al. provide more direct formulations for reciprocal collision avoidance of robots with car-like dynamics [36, 37]. Recently, more general approaches for reciprocal collision avoidance have been introduced: The approach of Alonso-Mora et al. [36] generalizes AVO and provides a formulation for robots with arbitrary-degree integrator dynamics (i.e. omnidirectional control of acceleration, jerk, snap, etc.). Reciprocal LQR-Obstacles [40] allow for collision avoidance of robots with arbitrary linear dynamics.

While the above approaches have successfully been tested in simulation, and some have been applied to collision avoidance for animated characters in virtual environments and games [45], few reciprocal collision approaches have been applied on real robots. HRVO, ORCA, and AVO [44, 35, 46] have been applied to iRobot Create robots with differential-drive dynamics, but with centralized sensing where a single sensor observes the environment and broadcasts the observed robot positions and velocities to all robots. To the best of our knowledge, reciprocal collision avoidance in 3-D has not been applied to real robots. Furthermore, neither 2-D nor 3-D RCA has been experimentally verified on independent agents with distributed, on-board sensing, where each robot perceives its surroundings by itself.

Other, mostly centralized approaches to collision avoidance, i.e. approaches where motions of robots are centrally coordinated, have been successfully applied to real robots. The method of [47] uses a centralized velocity obstacle (VO) occupancy map to perform collision avoidance for quadrotor helicopters. Our work extends this method to independent quadrotors with local sensing. The work in [48] shows a decentralized method for linear aircraft dynamics. A potential field implementation has been shown in simulations to work for nonholonomic agents, specifically fixed wing aircraft dynamics [49]. Like potential

field methods, di-polar navigation functions have been shown in simulations to guarantee collisions-free paths for aircraft agents in the presence of centralized sensing uncertainty [50].

Not only is our independent method easier to implement on large numbers of robots, but we believe that the ORCA method offers other important advantages. Our experiments show that other VO algorithms can be transitioned to methods using on-board sensing. ORCA improves on many of these other non-VO methods because of its simplicity. This is especially apparent when compared to potential field applications that require the tuning of many parameters to operate reasonably. Based upon our results explained in Section 4.3.2, we believe that ORCA and its extensions will be used prominently in real-world applications for collision avoidance.

Important specifically to on-board, real-time sensing is the addition of noise to the velocity obstacle model. In [43], the robots are enlarged by their one-sigma uncertainty ellipse to construct a conservative representation of the velocity obstacle. The work introduced in [51] incorporates sensing uncertainty by increasing the size of an agent as a function of the noise on the position signal. This work has been extended in the collision avoidance with location uncertainty (CLAU) method, which has been demonstrated on differential drive robots [52]. CLAU relies on a centralized model of the positions and velocities of agents using an uncertainty bounded by a particle filter.

Limitations in on-board sensing have been investigated in a variety of ways. Simulations of decentralized sensors with limited observation spaces have shown to not hinder collision avoidance given dynamic constraints [53]. This topic is further explained for holonomic vehicles with bounded velocity in [54].

CHAPTER 3

DEVELOPMENT OF AN EXPERIMENTAL AERIAL ROBOTICS LABORATORY

3.1 The Algorithmic Robotics Lab

3.1.1 Physical Lab Space

We established the Algorithmic Robotics Laboratory during 2012 to serve as an experimental aerial robotics environment in which to implement novel algorithms onto physical hardware (Figure 3.1 and 3.2). The flight environment is composed of a 20ft x 20ft netted cage designed with the assistance of Ultra Truss corporation. This netting is used to keep the robots and researchers safe during experimentation. The netting configuration differs from many other aerial-only labs in that it does not feature a net over the floor. Early on in the development of the ARL, we opted to use a hard foam flooring to keep open the option of ground robots.

3.1.2 Motion Capture

Motion capture is provided by eight V100:R2 cameras and Natural Point Tracker Tools software [55]. Each set of four cameras is linked to a Windows computer with a hub that aggregates all the video feeds. The cameras are limited to a small field of view, which complicates their use in the netted enclosure. Motion capture works by illuminating reflective spherical markers with a flood of infrared light, and then imaging said markers with a complementary metal —oxide semiconductor (CMOS) camera. Multiple cameras are used to triangulate the position of the motion capture markers in the environment. Depending on the workspace for the robots in a given experiment, camera reconfiguration is often needed to optimize both capture volume and resolution.

Camera arrangement for aerial work is shown in Figure 3.3 and Figure 3.4. A visualization of the maximum possible capture volume is shown in Figure 3.5. Although this visualization overestimates the ability of the Natural Point cameras, the representation

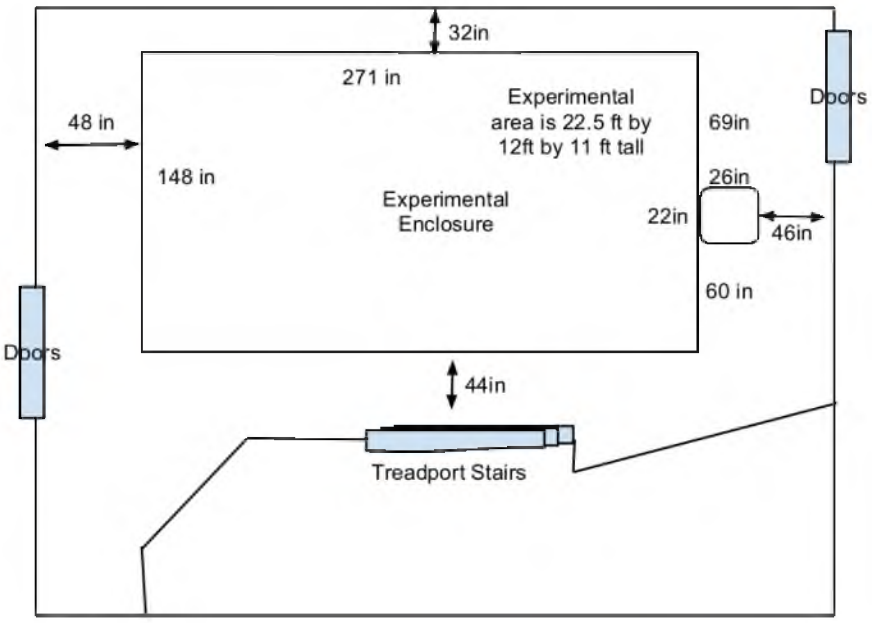


Figure 3.1. Top down view of the Algorithmic Robotics experimental environment.

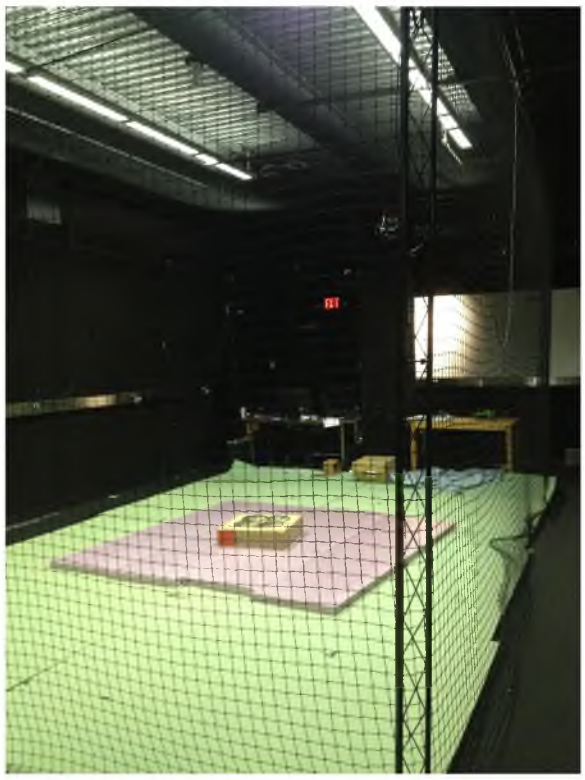


Figure 3.2. Photo of the Algorithmic Robotics experimental enclosure.

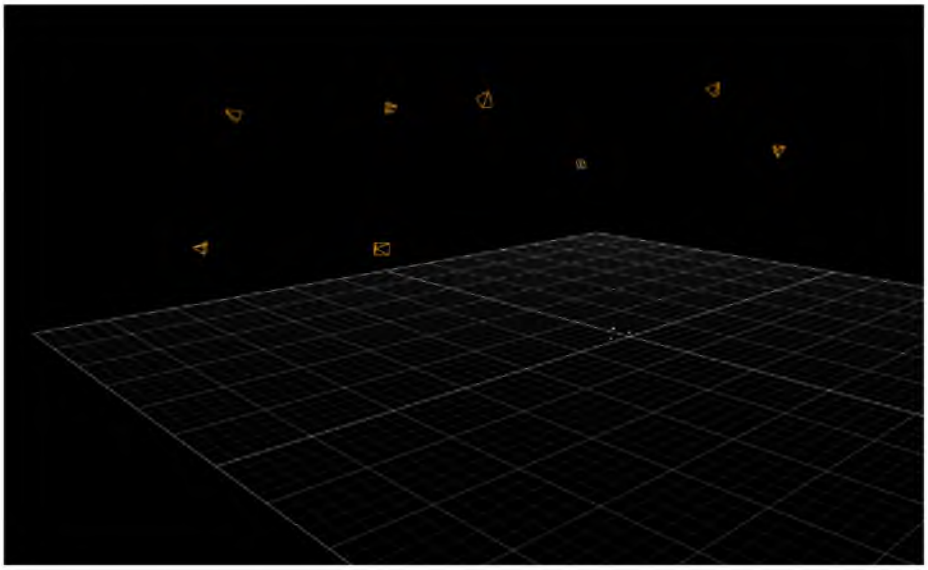


Figure 3.3. Visualization of camera layout.

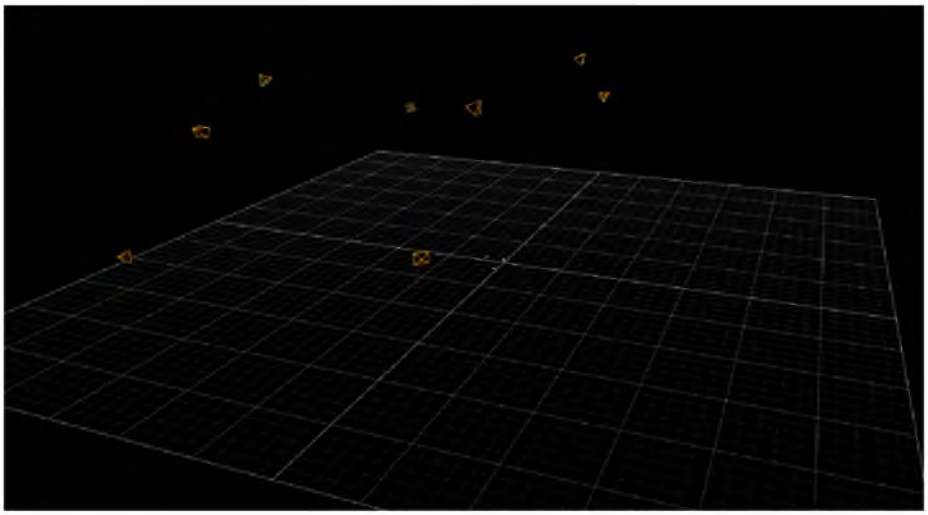


Figure 3.4. Second visualization of camera layout.

speaks to the number of cameras overlapped on the same volume. More accurate position estimates are given by reconstructions from more cameras.

The orientation of the cameras is a crucial component to the overall accuracy of the motion capture system because markers are not able to be resolved as circles once they become too small. Given that a target may be more than twenty feet away, extensive tests with various sized markers were completed to determine the effective minimum marker size. A static $\frac{1}{4}$ inch marker can only be resolved at twelve feet. A static $\frac{1}{2}$ inch marker can be resolved at a maximum of fifteen feet. Past these distances, the motion tracking software is unable to resolve the round nature of the marker from the pixelated image. When the markers move, this maximum resolving distance becomes much closer. In addition, the resolution of the position estimate becomes worse with smaller markers. This testing concluded that a relatively large diameter marker was needed. Figure 3.6 shows the largest available spherical $\frac{3}{4}$ inch diameter marker that was used for the experiments. This marker was able to be resolved at up to nineteen feet, and at about ten feet confidently in motion. An image of the camera output of these three markers at twelve feet can be seen in Figure 3.7.

After calibration, the Tracker Tools software is able to determine the position of the markers with an error of approximately one millimeter [56]. Robots in the environment are equipped with up to seven infrared lights or reflective markers. These markers can be tracked in position and orientation as a single rigid object. Using Virtual-Reality Peripheral Network (VRPN), we stream the robot position and orientation to a Linux terminal for processing and data logging. Preliminary tests and work by other groups [11] show that the VRPN link induces a latency of about ten milliseconds in the data, although this delay is inconsistent.

3.2 Robot Operating System

ROS features a distributed system in which programs exchange variables based upon a peer-to-peer messaging system and remote procedure calls. A core ROS node manages the interconnectivity between the programs, where both local and distributed variables are communicated over TCP and UDP. Programs written in Java, C++, and Python are able to be simultaneously used together. Standardized variables for common message types are offered to give a basic platform for communication, although more complex variable packages can be shared between programs.

While a Linux ROS architecture was finally decided upon for the software environment, many other robotics oriented environments are used for research. The most popular other

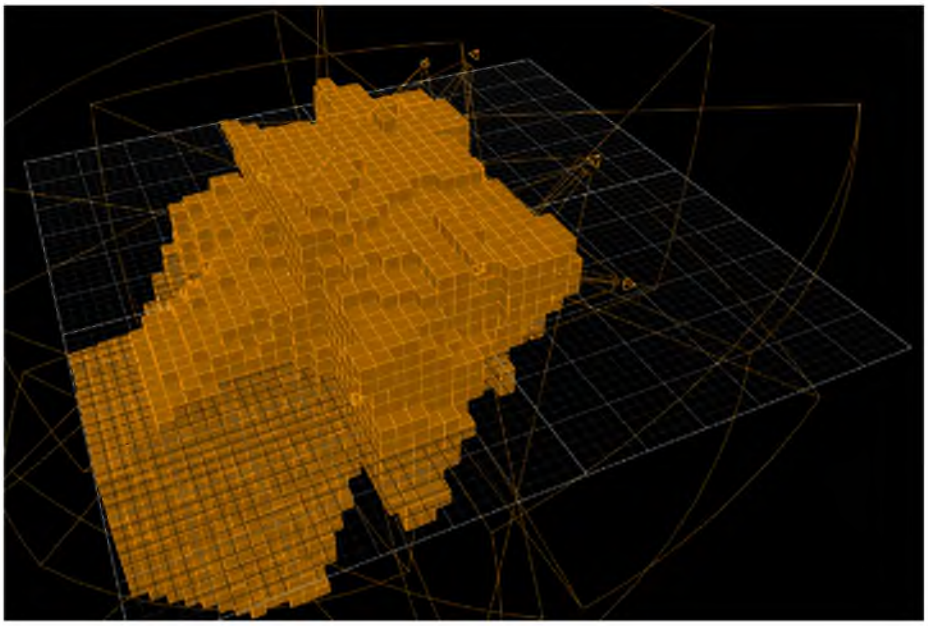


Figure 3.5. Visualization of camera capture volume.

Figure 3.6. 0.75 in OD reflective motion capture marker.

Figure 3.7. Image of the $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$ motion capture markers at twelve feet.

framework is Fawkes [57]. Fawkes is a tightly regulated component nondistributed architecture focusing on a small number of message types. It features many of the capabilities of the ROS system except it focuses on threaded systems in which data is passed around in shared memory.

Another system, open robot control software (OROCOS) [58], is also widely accepted. ORCOCOS is a C++ library specializing in real-time implementations of kinematics and Bayesian filtering. It is used generally for motion planning applications.

A variety of other software packages are implemented in either distributed or stand-alone robotic systems. Player Stage Gazebo is generally used for robotics simulation [59]. Microsoft Robotics Developer Studio [60] features visual studio integration and Kinect sensor [61] modeling. CLARAty [62] is a robotic infrastructure and compilation of navigation, path planning, vision, and other algorithms maintained by NASA.

We chose to use ROS over these other systems because of the research communities acceptance of the ROS structure and the increased freedom ROS gives to its program. As of the writing of this document, ROS has gained a foothold as the standard for robotics research, leading to many of the mentioned architectures being integrated into ROS.

3.3 Experimental Architecture

The Algorithmic Robotics Lab computing consists of two Intel i7 desktop computers, as described in Table 3.1. The first of these machines operates in Microsoft Windows and is responsible solely for running the tracking software for the motion capture and streaming said data to the second machine. The second computer performs all the other necessary functions of the lab, including but not limited to running the experiment-specific software and communicating with the robot(s). The second system runs Ubuntu 11.10, with a Robot Operating System (ROS) shell. Both of these machines were custom built on site for the laboratory with a focus on building the most capable computers for under one thousand dollars. This setup is shown in Figure 3.8.

Since the laboratory was designed with multiple robots in mind, a variety of communication methods were developed for each specific robot. From the desktop, many different communication architectures and hardware types are used to distribute control signals and receive data back to the central ROS system. Each of the following robots have been developed for or tested in the laboratory, where a full list of ROS robots is available on the wiki [63]. The AR.Drone [64] robot (Figure 3.9) runs a proprietary Linux distribution that takes inputs over an 802.11n wi-fi link. The Turtlebot robot [65] (Figure 3.10), running ROS, uses an 802.11n wi-fi signal, which is then sent over serial to an Atmel

Table 3.1. Desktop Computers used in the ARL

Type	Model
Case	Thermaltake V3 Black Edition VL80001W2Z
CPU	Intel Core i7-2600 Sandy Bridge 3.4GHz
PSU	Rosewill HIVE Series HIVE-650 650W
RAM	G.SKILL Ripjaws X Series 4GB (4 x 2GB) 240-Pin DDR3 1600
CD	SAMSUNG CD/DVD Burner SH-222AB
VC	HIS H675FS1G Radeon HD 6750 1GB 128-bit DDR3
MB	GIGABYTE GA-P67A-UD4-B3 LGA 1155 ATX Intel Motherboard

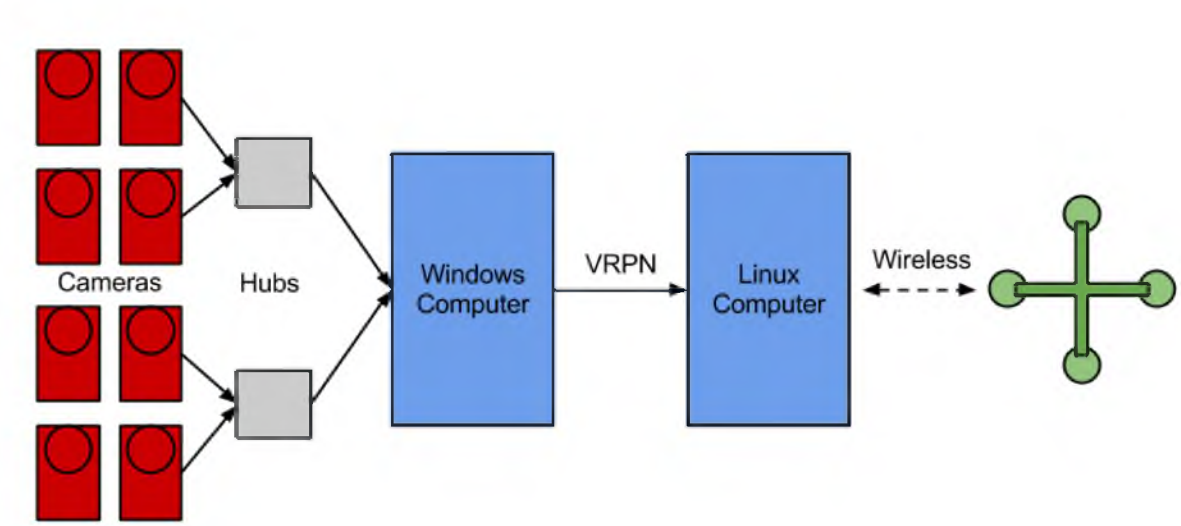


Figure 3.8. Graph of the ARL systems.

Figure 3.9. Photo of AR.Drone robot.

microcontroller. The Sphero robot [66] (Figure 3.11) uses a wireless 802.15.1 Bluetooth connection at 2.4Ghz. The larger custom quadrotor (Figure 3.12) is linked to the ROS system using an Xbee XB24-AWI-001-ND wireless 802.15.4 radio operating at 2.4Ghz [67]. The smaller of the custom quadrotors (Figure 3.13) relies on a pulse position modulation (PPM) signal generated from an Atmel microcontroller and sent via a hacked RC aircraft transmitter. While the specifics for each of these applications is outside the scope of the thesis, the code for each of these applications can be found on our repository [68]. Both of the custom quadrotors, built in-house for the use of ARL, are described in Appendix C.

3.4 Attitude Estimation and Control

When flying a quadrotor to a specific position in the ARL environment, visual servo control methods [69] were used in a motion capture context to fly the robot within the volume. Due to the drift and absolute error in the on-board accelerometer and gyroscope, dead reckoning is not enough to maintain a desired position with constant steady-state error. The high level position control operates on top of the local controller, which actuates the motors of the quadrotor. As stated in Equation 2.3, the orientation of a quadrotor is coupled with its acceleration due to thrust vectoring.

The attitude of the aircraft as referenced from the static world orientation is represented by a rotation matrix. The world coordinates are configured so the $\hat{z}$ direction points directly opposite to that of gravity and the $\hat{x}$ direction is set arbitrarily according to the experimental setup. This matrix can represent the rotation from the world frame to the local frame, or vice versa. Orientation estimates are stored as a direction cosine matrix (DCM). A DCM is an orientation parameterization that provides a rotation matrix with no singularities (Equation 3.1) since it only relies on the cross product of world and local vectors. This same DCM can also be calculated as the product of rotation matrices about roll, pitch, and yaw, as seen in Equation 3.2. This Tait Bryan rotation matrix is affected by gimbal lock and thus has singularities near ± 90 deg of roll and pitch when measured from a hover state.

$$R = \begin{bmatrix} \cos\theta_{xX} & \cos\theta_{xY} & \cos\theta_{xZ} \\ \cos\theta_{yX} & \cos\theta_{yY} & \cos\theta_{yZ} \\ \cos\theta_{zX} & \cos\theta_{zY} & \cos\theta_{zZ} \end{bmatrix} \tag{3.1}$$

$$R = \begin{bmatrix} \cos\phi\cos\psi & -\cos\phi\sin\phi & \sin\phi \\ \cos\theta\sin\psi + \cos\phi\sin\theta\sin\phi & \cos\theta\cos\psi - \sin\theta\sin\phi\sin\psi & -\cos\phi\sin\theta \\ \sin\theta\sin\psi - \cos\theta\cos\psi\sin\phi & \cos\psi\sin\theta + \cos\theta\sin\phi\sin\psi & \cos\theta\cos\phi \end{bmatrix} \tag{3.2}$$

Where:

x, y, z implies the local frame

Figure 3.10. Photo of Turtlebot robot.

Figure 3.11. Photo of Sphero Robot. Orbotix, Inc. All rights reserved. Reproduced according to [3]

Figure 3.12. Photo of custom quadrotor.

Figure 3.13. Photo of the smaller quadrotor built exclusively for the ARL.

X, Y, Z implies the world frame

θ is the "pitch"angle from the world $\hat{x}$ to the local $\hat{x}$ measured about the local $\hat{y}$

ϕ is the "roll"angle from the world $\hat{y}$ to the local $\hat{y}$ measured about the local $\hat{x}$

ψ is the "yaw"angle from the world $\hat{x}$ to the local $\hat{x}$ measured about the local $\hat{z}$

3.4.1 Sensors

The quadrotors used in the laboratory, like most quadrotors [70] [18] [71], rely on two sensors to estimate orientation. The first on-board sensor is a rate gyroscope and the second is an accelerometer. An accelerometer inherently measures acceleration, which can provide orientation by measuring the only constant acceleration on the aircraft: the gravity vector. This orientation observation works in a semistatic dynamics model very well, where attitude changes are a results of only small accelerations. Once dynamic forces begin to act on the craft, the semistatic assumption is violated and the sensor's measurements cannot be used directly as a measure of orientation. Rate gyroscopes, as the name implies, measure the change in orientation, requiring numerical integration before they can be used to estimate orientation. This measurement is also suspect due to the tendency for rate gyros to drift.

While orientation could be estimated by using only an accelerometer, or rate gyroscope, each of these methods alone provides inaccurate attitude estimates. The best estimate of the quadrotor's orientation requires the fusion of all the on-board sensors. While a variety of filtering methods exist, we choose a complimentary filter because it has low computational requirements [72]. This filter is used commonly used for this application in embedded systems, again due to the simplicity and minimal computational requirements. The important decision in the creation of such a filter is the time constant used for each of the standalone internal filters.

$$\tau = \frac{a * dt}{1 - a} \tag{3.3}$$

$$x_t = (1 - a) * (x_{t-1} + x_{gyro} * dt) + (a) * (x_{acc}) \tag{3.4}$$

Where:

dt is the time between data samples.

a is the smooth factor.

x_{t-1} is the previous estimate.

x_t is the current estimate.

x_{gyro} is the measurement from the gyro.

x_{acc} is the measurement from the accelerometer.

τ is the time constant of the filter.

This implementation of the complimentary filter can be seen as the fusion of a low-pass filter acting on the accelerometer signal, which cleans up fast vibrations from the motors, and a high-pass filter, which conditions the rate gyroscope signal against drift [73]. We tested a variety of time constants from a tenth of a second to five seconds, finally deciding upon a time constant of half a second. With our chosen time constant, Equation 3.4 can be understood as trusting the accelerometer over time periods past half a second and the gyro over smaller time periods [74]. A vibrational isolator, or mechanical filter, is also used to attenuate high-frequency vibrations from each of the sensors, although its effects were not included in the model.

3.4.2 Controls

The quadrotor's motor inputs are defined by a proportional, integral, and derivative (PID) controller. This controller acts on the newest state estimates and computes motor inputs at 400Hz on an ATmega2560 microcontroller [75] running at 16MHz [76]. Strapdown controllers are designed to keep the aircraft flying stably on the desired heading [77] and are often referred to as attitude and heading reference systems (AHRS) or inertial navigation systems (INS). Strapdown controllers are defined as both an attitude estimator and the control loops that provide inputs based upon those estimates and the desired heading. The velocity of the craft is controlled as a function of orientation (Equation 2.3), by actuating the aircraft's inputs, i.e. the quadrotor's four motors, according to the desired attitude and current estimated attitude (Equations 3.5 and 3.6). Roll, pitch, and yaw are each controlled independently according to:

$$\tilde{x} = \hat{x}_{desired} - \hat{x}_{estimate} \tag{3.5}$$

$$u = K_p * \tilde{x} + K_d * \dot{\tilde{x}} + K_i * \int \tilde{x} \tag{3.6}$$

Where:

$\tilde{x}$ is the error in the axis' measurement.

$\hat{x}_{desired}$ is the desired roll, pitch, or yaw.

$\hat{x}_{estimate}$ is the instantaneous estimated roll, pitch, or yaw.

u is the control input.

K_p, K_d, and K_i are gains.

An off-board high-level controller works exclusively on the position and velocity of the quadrotor, operating on much the same principle as the local controller. Since the local controller acts as a bridge between the orientation and the motor inputs, the high level

controller is left to act upon the velocity of the aircraft as a function of the orientation. In our laboratory setup, this controller runs off-board on a desktop computer that gathers position information from the motion capture computer. This high-level controller is a PD controller with gravity compensation, see Equation 3.10. Basic gravity compensation is determined experimentally by increasing the total thrust until hover is achieved, while it varies as a function of orientation.

$$\tilde{x} = \hat{x}_{desired} - \hat{x}_{estimate} \tag{3.7}$$

$$\tilde{y} = \hat{y}_{desired} - \hat{y}_{estimate} \tag{3.8}$$

$$\tilde{z} = \hat{z}_{desired} - \hat{z}_{estimate} \tag{3.9}$$

$$u_{thrust} = K_p * \tilde{z} + K_d * \dot{\tilde{z}} + \hat{G} \tag{3.10}$$

$$u_{roll} = K_p * \tilde{y} + K_d * \dot{\tilde{y}} \tag{3.11}$$

$$u_{pitch} = K_p * \tilde{x} + K_d * \dot{\tilde{x}} \tag{3.12}$$

$$\hat{G} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{3.13}$$

Where:

$\tilde{x}$ is the error in global x position.

$\hat{x}_{desired}$ is the desired global x position.

$\hat{x}_{estimate}$ is the instantaneous estimated global x position.

$\tilde{z}$ is the error in global z position.

$\hat{z}_{desired}$ is the desired global z position.

$\hat{z}_{estimate}$ is the instantaneous estimated global z position.

$\tilde{y}$ is the error in global y position.

$\hat{y}_{desired}$ is the desired global y position.

$\hat{y}_{estimate}$ is the instantaneous estimated global y position.

u_{thrust} is the control input for the total thrust.

u_{roll} is the control input for the left and right motor.

u_{pitch} is the control input for the front and back motor.

K_p, K_d, and K_i are gains.

g is a quantification of thrust needed to hover the craft.

Pursuant to the quadrotor dynamics as described in Section 2.3, when the craft pitches forward, it creates an acceleration in that direction. Controls for pitch and roll are added

to the back and left motor, respectively, and subtracted from the front and right motor, respectively. While the roll and pitch of the quadrotor are necessary for the actuation in global $\hat{x}$ and $\hat{y}$ directions, the yaw angle does not affect these movements. Since our desire is to fly the craft without rotating about the quadrotor's local $\hat{z}$, the desired hover position was rotated as to align with the quadrotor's instantaneous $\hat{x}$-$\hat{y}$ coordinates. This allows the aircraft to be flying without direct yaw control. This method was used in practice allowing for the yaw to drift, while the global control changed the inputs based upon the instantaneous yaw measurement gathered from the motion capture.

3.5 Prototype One

The first experimental architecture used in the lab consisted of primarily one program handling the controller inputs, PID controller, filtering, and output to the robot. We designed this system as a finite state machine (FSM) due to concerns for latency and operational speed, using plug-in classes for additional functionality. Because of the need for the classes to be similar, robots were limited to functioning within a narrow framework concerning a six degree of freedom control signal. The state machine is formatted much like what will be seen in prototype two, but does not take advantage of modular nodal structure, and thus does not allow for the quick swapping of nodes. Plug-ins can take the form of robot drivers, controllers, and trajectory inputs. The structure of the code is as follows according to Figure 3.14 and Figure 3.15. An example of the visualization methods can be seen in Figure 3.16. We found that debugging and developing additional functionality became increasingly time-consuming the larger the architecture became.

3.5.1 Prototype One Validation

The first experimental architecture was tuned and tested with position tracking of a quadrotor. Motion capture markers were installed on the rotorcraft so that its position could be monitored via motion capture. A PID controller was used to apply controls to the robot so that it hovered at a predetermined point in the flying arena. The system was run through a variety of tests concerning steady-state stability and disturbance rejection. Graphs for the experiments are shown in Table 3.2, and can be found in Appendix B. These experiments are not analyzed in detail because they were used merely as a validation of each of the lab's systems. While the finite state machine was suitable for these experiments, the testing concluded that the software would become too large to effectively manage a complex experiment.

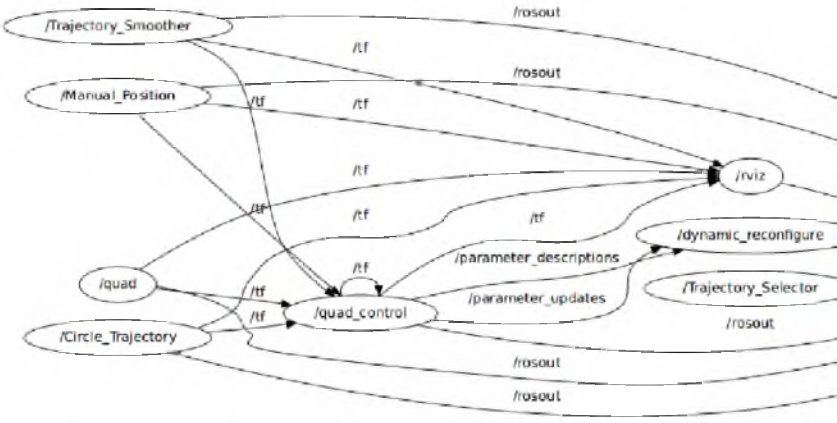
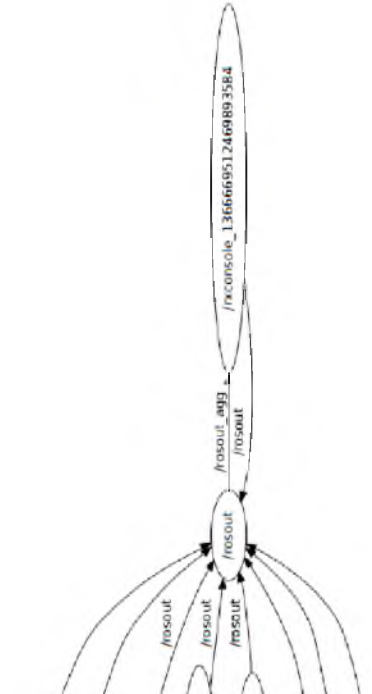


Figure 3.14. Graph of prototype one's ROS structure.


/rosout_agg
/rosout
/rosout
/rosout
/rosout
/rosout
/rosout

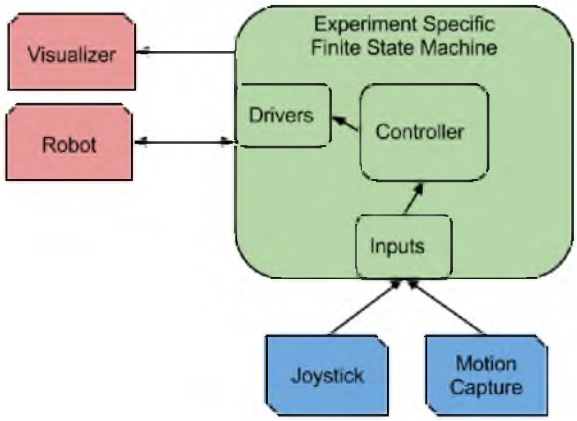


Figure 3.15. Graph of prototype one's architecture.

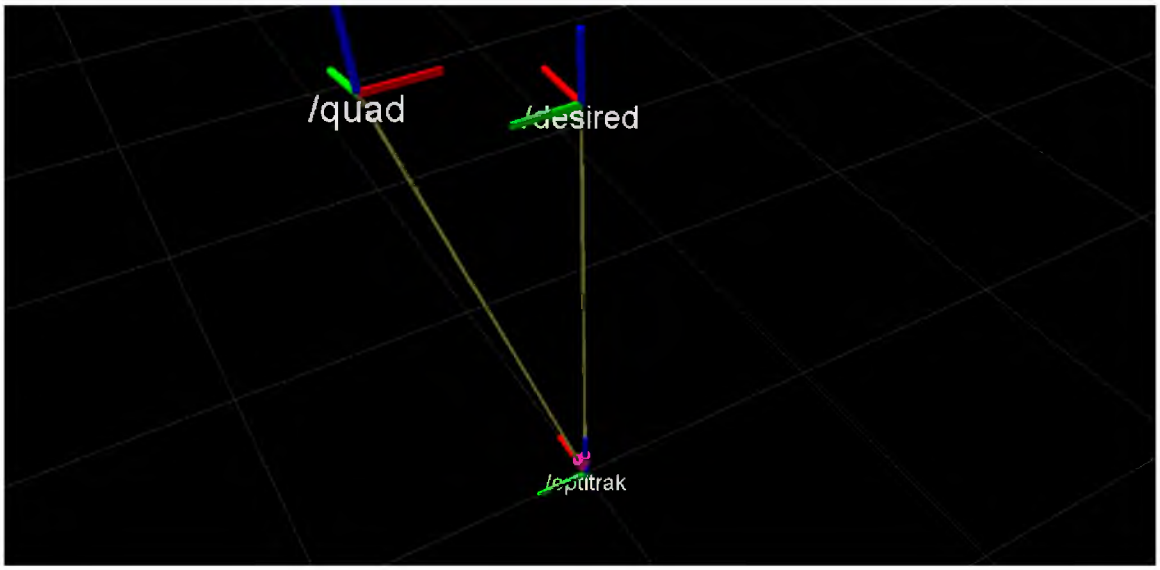


Figure 3.16. Example of prototype one's visualization.

Table 3.2. Prototype One Tests

Experiment	Input	Disturbance Direction
1	1m X Step from hover	N/A
2	1.5m Z Step from ground	N/A
3	Hover	30cm displacement in the global X and X-Y
4	Hover	20cm displacement in the global Z

3.6 Prototype Two

After running experiments using the first experimental architecture, we desired to shift from its monolithic nature to a more distributed architecture. This involved separating the code into smaller, more manageable pieces with the desire of making an architecture that could evolve faster to accommodate different styles of experimentation and robot types. We found that this new architecture allows for easy debugging and observation of each subsection of an experiment. By limiting the scope of each node, the likelihood of using existing ROS nodes increases, shortening the experimental cycle. For example, the widely used Xbox controller [78] node allows one to quickly add a human interface to an experiment, where otherwise construction of such a functional driver set could take weeks.

Using nodes distributed by other researchers allows ARL researchers to focus on parts of the code pertinent to the novel ideas in the experiment. Most importantly, isolating the new experiment-specific sections of the code allows for quick prototyping and bug removal. Focusing on human robot interface techniques streamlined the system and made it easier to use. These node types are discussed in detail below and demonstrated in Fig. 3.17.

ROS nodes have the ability to be run like standard C++ programs from the terminal or to be started in bulk using XML launch files. The launch files also allow one to remap the messages coming in and out of the nodes, by putting a portion of or the entire experiment into a *name space*. This allows multiple sets of robots to be started and implemented into the larger experimental framework with little effort, allowing easy transitions to centralized or distributed formats. Architecture reconfiguration takes place in global launch files which call subsections of larger launch files; these launch files then call the desired nodes. Global parameters such as the experimental loop time and the internet protocol address (IP) of the robots and VRPN servers are distributed using launch files. These programs are generally run at 100–200 Hz. A global setting is used to manage this speed, which is a constant across each of the programs used in the modular setup. This timing is controlled using the ROS loop commands and features a measured error of less than one percent. A global launch file is given in Appendix A.1 and an experimental launch file in given in Appendix A.2.

3.6.1 Node Styles

The following is a description of the node styles developed for use in the second prototype's architecture, where an example connectivity graph is shown in Figure 3.17 and Figure 3.18. Nodes are designed to use well-accepted variable types to assist in interconnectivity. A vector composed of three elements is an example of these standard variables. These standard message types are well-defined variables decided upon by the ROS community to

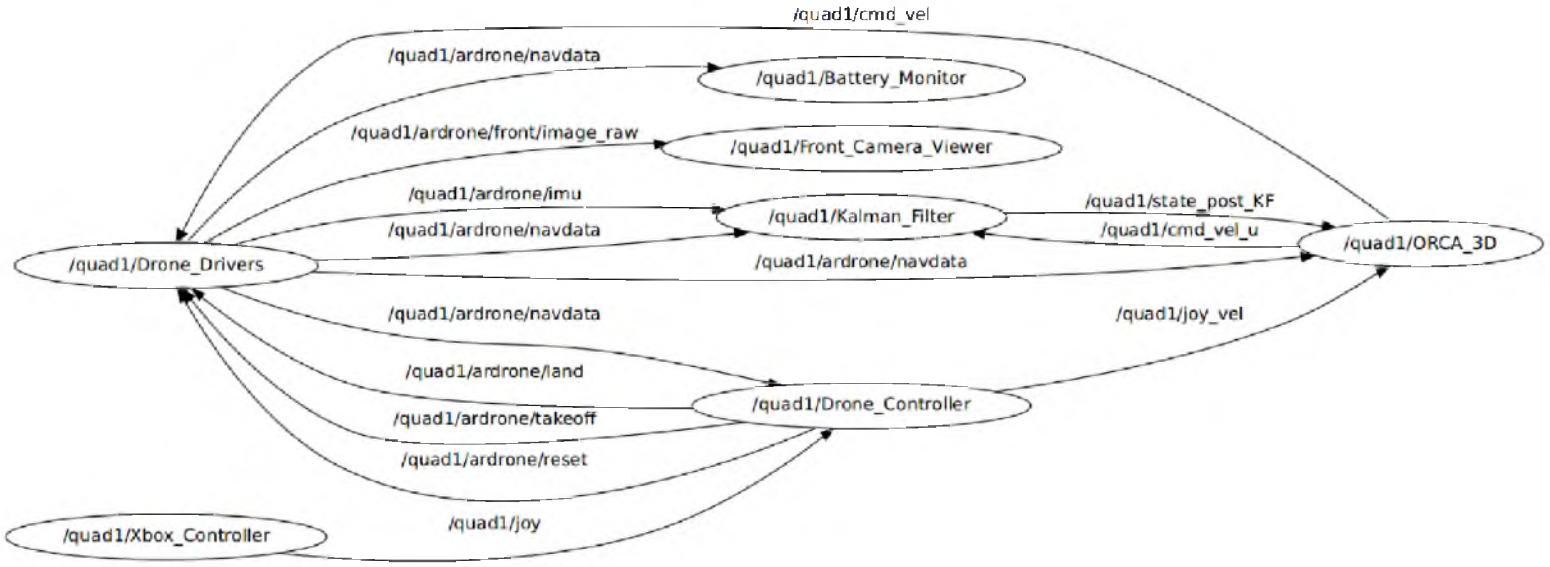


Figure 3.17. Example of prototype two experimental ROS architecture.

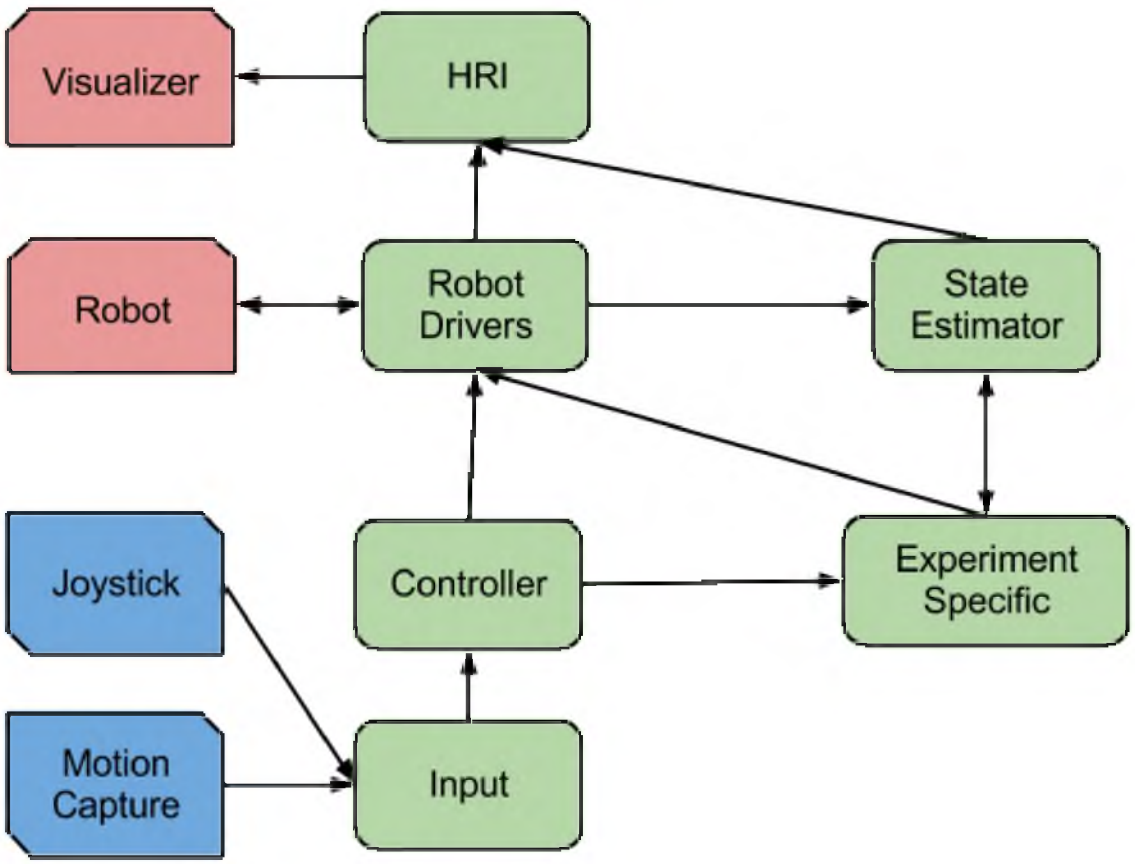


Figure 3.18. Graph of prototype two architecture.

pass around data between nodes. Custom message types are often used with robot drivers, necessitating the use of the controller node type to dissolve the needed information to each of the more specific node types.

3.6.1.1 Input nodes

Human interface devices, and other means of inputing high-level information into the experiment, are conducted within an input node. These nodes are used to provide human interaction with an experiment, independent of the interface. Drivers for human input devices are located within these nodes, as to not clutter the experimental framework further. With this architecture, a wired Xbox joystick can be swapped out for a motion capture controller within minutes. Changing the experimental robot often involves changing the output space's degrees of freedom (DOF), which is reflected in the size of the control message streamed from the joystick.

Inputs:

Human Interface Devices: joysticks (such as Xbox controllers), haptic interfaces, keyboard/mouse, etc. (USB interface)

Motion Capture: position/orientation of the robot in the environment, position/orientation of a human actuated virtual robot or controller (VRPN)

Outputs:

Position and Orientation: six DOF orientation message composed of 2 vectors with three elements for linear and angular geometries (geometry_msgs/Twist.msg)

Joystick: one array of joystick axises, floats, and one array of integers for buttons (sensor_msgs/Joy.msg)

3.6.1.2 Controller nodes

Controller nodes are designed to be the managers of the experiment, especially focused on robot controls. These nodes take data from the input nodes and distribute the pertinent data to other nodes. Controller nodes are the switchboard that allows easy interconnectivity between the other, more specialized, node styles used in this experimental framework.

Pertinent to the robot, controller nodes output the necessary inputs as expected by the robot driver nodes. Examples of the types of output are command inputs, robot initializations, and mode changes such as takeoff, landing, and resets. The reason that

control nodes must manage the robot on this higher level is to allow driver nodes to be independent of the experimental setup.

Inputs:

Robot Commands: velocity or orientation message

Robot Mode: binary messages or custom messages for changing on-board gains

Robot Specific Outputs:

AR.Drone: 4 DOF *twist* message

Sphero: 2 DOF velocity array inputs

Turtlebot: 2 DOF *twist* message

Custom Quadrotor: 3 DOF velocity message

3.6.1.3 Driver nodes

Driver nodes are responsible for taking the command inputs from the controller nodes and properly packaging them for the robot(s). Packet management is conducted within the driver node because, generally, robot controls need to be formated before the signal is sent to the hardware. Serial connectivity and wi-fi are the main methods of communication with the ARL's robots. Often robots emit a state message back into the experimental framework for the sake of robot state feedback.

Robot Specific Outputs:

AR.Drone: custom *Navdata* message consisting of entire state

Sphero: custom message consisting of LED color, velocity

Turtlebot: custom message with battery information, current mode, last message received

Custom Quadrotor: heartbeat

Robot-Specific Communication Methods:

AR.Drone: Packets are distributed over wi-fi using a proprietary format.

Sphero: Controls are encoded into proprietary packets and transfered over Bluetooth.

Turtlebot: The turtlebot uses a laptop sharing the ROS workspace over wi-fi so that it may get the needed control messages. These messages are then transfered over serial RS-232 to a wired ATMEL microprocessor, which implements the digital motor controls to the turtlebot's microcontroller.

Custom Quadrotor: Data are transfered from the driver node over serial to an Xbee radio. On-board the quadrotor, a paired Xbee and ATmega2560 process a PWM signal for the AHRS.

3.6.1.4 HMI nodes

Human machine interface nodes specialize in displaying data, so that the researcher is shown only important information. Often in robotics work, researchers are burdened with large data streams featuring information not directly pertinent to the experiment. Displays are generally composed of built-in ROS functionality such as message graphing and visualization techniques supported by OpenCV [79]. Instantaneous graphing is also used to show data, for example position error on one axis and the control input on another.

These abilities were used to full effect in the collision avoidance work, Chapter 4, by streaming the instantaneous state estimate and the full Kalman filtered state. Images from the front-facing camera on the quadrotor can also be displayed. An example can be seen in Fig. 3.19. Two other configurations are used in practice: global mode streams from the quadrotor camera, flight environment camera, and rigid body motion capture are shown in visualizer (Fig. 3.20). Controls mode streams the desired point or trajectory and the current attitude of the robot.

HMI Modes:

Global: display top level camera and motion capture information

State: display on-board camera and visualize estimated experimental states

Experiment/Control: visualize desired and current state

3.6.1.5 State estimator nodes

While not explicitly necessary in some experiments, a node specializing in filter techniques is useful for keeping a universal state shared between all of the nodes. Filtering techniques such as complimentary or Kalman observers have been used to clean up data before it is sent to the controller and HMI nodes. While this node appears to work

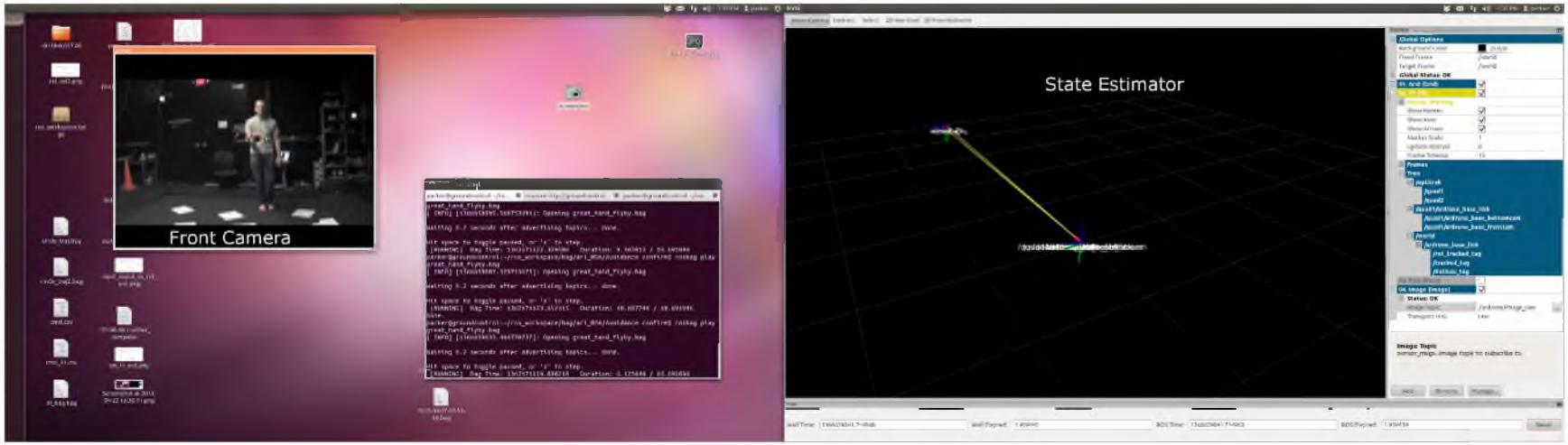


Figure 3.19. Screen shot of desktop configuration displaying the state estimate display.

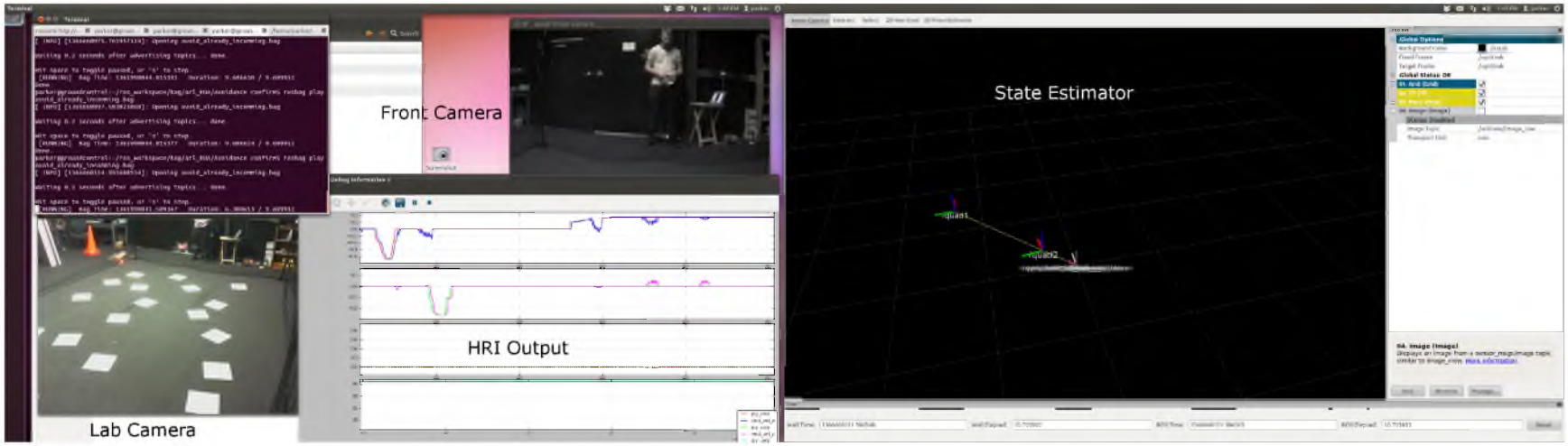


Figure 3.20. Screen shot of desktop configuration displaying the global experimental display.

specifically for centralized systems, multiple instances of this portion of the architecture can be used for distributed experimental systems.

3.6.1.6 Experiment-specific nodes

The novel section of the experiment is built in these nodes. Examples of these include collision avoidance, mapping, and global position/trajectory controllers. As seen in Chapter 4, this node ran a collision avoidance algorithm that modified human controls when an imminent collision was detected between two AR.Drone 2.0 robots [80].

CHAPTER 4

3-D RECIPROCAL QUADROTOR AVOIDANCE

This chapter discusses the work submitted for publication, *3-D Reciprocal Collision Avoidance on Physical Quadrotor Helicopters with On-Board Sensing for Relative Positioning* [80]. This paper was collaboratively written by Parker Conroy, Daman Bareiss, Matt Beall, and Dr. Jur van den Berg. Parker Conroy, as the author of this thesis, is responsible for the development of the Kalman filter, all software and hardware work concerning the quadrotor robots, the ROS framework, developing the understanding of exponential convergence and the reciprocal dance, and any other work not including the transition of ORCA into the ROS framework. Daman Bareiss is responsible for the work in which ORCA was reconditioned from its native Visual Basic simulation form into a ROS node, in addition to assisting with the mathematical ideas developed herein. Matt Beall assisted with running the experiments, created the images given in the research paper, and formatted the conference video. ORCA itself was conceived by Dr. Jur van den Berg, who advised each aspect of this research.

4.1 Reciprocal Collision Avoidance

Applying RCA on real robots poses a number of unique challenges. In particular, the ORCA framework relies on perfect symmetry (i.e. a pair of robots observe exactly the same relative position and velocity with respect to each other) and perfect reciprocity (i.e. both robots take half the responsibility of avoiding collisions) to guarantee collision avoidance. In settings where each robot uses its own on-board sensing, the symmetry assumption is inherently violated due to sensor noise. In addition, the ORCA framework assumes that robots can adopt any velocity instantaneously. While extensions exist for robots with more complicated dynamics, real robots will always deviate from the expected behavior as a result of external disturbances or modeling error.

This chapter describes the experimental validation (Figure 4.1) of a 3-D implementation of ORCA [34] in GPS and motion capture denied environments on fully independent quadrotor helicopters that sense other robots using their own on-board sensors. To our knowledge, this work is the first to apply 3-D reciprocal collision-avoidance on real robots, and the first to apply reciprocal collision-avoidance on real robots with on-board decentralized sensing.

We both qualitatively and quantitatively analyze the effect the violation of symmetry, reciprocity, and dynamic assumptions has on the collision avoidance behavior of the robots, and we show that ORCA is mostly robust against these violations. In particular, our experimental results suggest that sensor noise can lead to *reciprocal dances*, even though not leading to collisions. This phenomenon is also observed in human motion, and our analysis suggests this is the result of asymmetry in the sensing of relative position and velocity among a pair of robots; one robot believes the robots should pass each other on the right while the other robot believes they should pass each other on the left. The result is that both robots choose a trajectory that is anew on a collision-course, which may lead to a repetition of the phenomenon until the sensing asymmetry breaks. This phenomenon is most likely to occur when the robots approach each other "head-on", in which case there is no obvious side to pass each other. We will theoretically analyze the nature and origin of reciprocal dances in RCA.

Also, our experimental results suggest that ORCA is robust against the violation of the assumed dynamics. The robots will exponentially fast converge to a collision free trajectory even when the robots are not able to instantaneously assume such a trajectory. The same mechanism is at play when the reciprocity assumption, i.e. the assumption that the other robot takes care of its share of the responsibility of avoiding collisions, is violated by an other robot. The robot will then converge to a collision-avoiding trajectory with an exponential rate.

We experimented with our framework on a pair of Parrot AR.Drone quadrotors in multiple environments where no external motion capture or GPS sensing were used. All the necessary sensing was preformed by on-board CMOS cameras. Vision algorithms were used to detect other quadrotors in the image frame of a forward-facing camera. As the crafts entered each other's detection envelope of the camera system, these noisy measurements were processed through a Kalman filter which estimated the desired relative position and relative velocity with respect to other quadrotors. In all of our experiments, the quadrotors were flown by human operators, and no collisions occurred even if the operators attempted to steer the quadrotors along malicious, colliding trajectories.

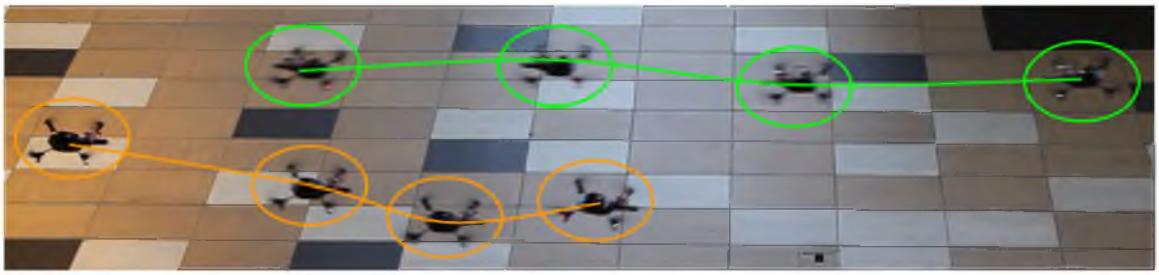


Figure 4.1. Two quadrotors are controlled along a straight-line trajectory by a user and our reciprocal collision avoidance algorithm corrected the control velocities to produce a collision-free trajectory.

We hypothesize that sensing uncertainty in reciprocal collision avoidance can lead to *reciprocal dances*. Reciprocal dances have a limited theoretical backlog of work, but a few researchers have investigated it in both human crowd dynamics and agent simulation. First mentioned in 1971 [81], this phenomenon was shown in human experiments concerning movement through bottlenecks in [82]. A crowd collision model agrees with the hypothesis that miscommunication between agents leads to the nonoptimal behavior [83]. We instead hypothesize that reciprocal dances are a direct result of asymmetry in the sensed relative positions and velocities by pairs of agents, and provide a theoretical explanation for the phenomenon in the context of RCA.

4.1.1 Optimal Reciprocal Collision Avoidance

The optimal reciprocal collision avoidance (ORCA) algorithm is at its core a velocity obstacles method. Given two agents i and j sharing a workspace, the velocity obstacle $\mathcal{VO}^{\tau}_{ij}$ of robot i with respect to robot j, is a geometric representation of the set of *relative* velocities that will result in a collision between robot i and j within τ time in the future. Each robot can be represented by a simple shape such as a disc of radius r. For robot i to create a velocity obstacle with respect to robot j, it must be able to determine *relative* position $\mathbf{p}_{ji} = \mathbf{p}_j - \mathbf{p}_i$ and the sum of the radii of the robots (see Fig. 4.2). A velocity obstacle $\mathcal{VO}^{\tau}_{ij}$ is defined as a volume or surface constructed with the size of the agents, their relative position, and relative velocity according to Figure 4.2. Given a velocity obstacle, a collision is imminent before time τ if

$$\mathbf{v}_{ij} \in \mathcal{VO}^{\tau}_{ij}, \tag{4.1}$$

where $v_{ij} = v_i - v_j$, is the current relative velocity.

Referring to Figure 4.2, ORCA determines the minimum change in relative velocity necessary to avoid collision represented by vector $\mathbf{u}$. The reciprocal aspect of the algorithm assumes each robot will take equal responsibility to avoid collision, therefore, each robot changes their velocity by at least half of the required change, thus changing the *relative* velocity by at least the full correction. Robot i would then change its velocity by at least $\frac{1}{2}\mathbf{u}$ such that $\mathbf{v}_i$ is in the halfplane through $\mathbf{v}_i + \frac{1}{2}\mathbf{u}$ perpendicular to $\mathbf{u}$. Symmetrically, robot j changes its velocity by at least $\mathbf{v}_j - \frac{1}{2}\mathbf{u}$

In all, a robot i requires the relative position, the relative velocity, and its own absolute velocity to construct the halfplane of valid new velocities with respect to each robot j. It then selects a new velocity from the intersection of all halfplanes. The concept is naturally extended to 3-D.

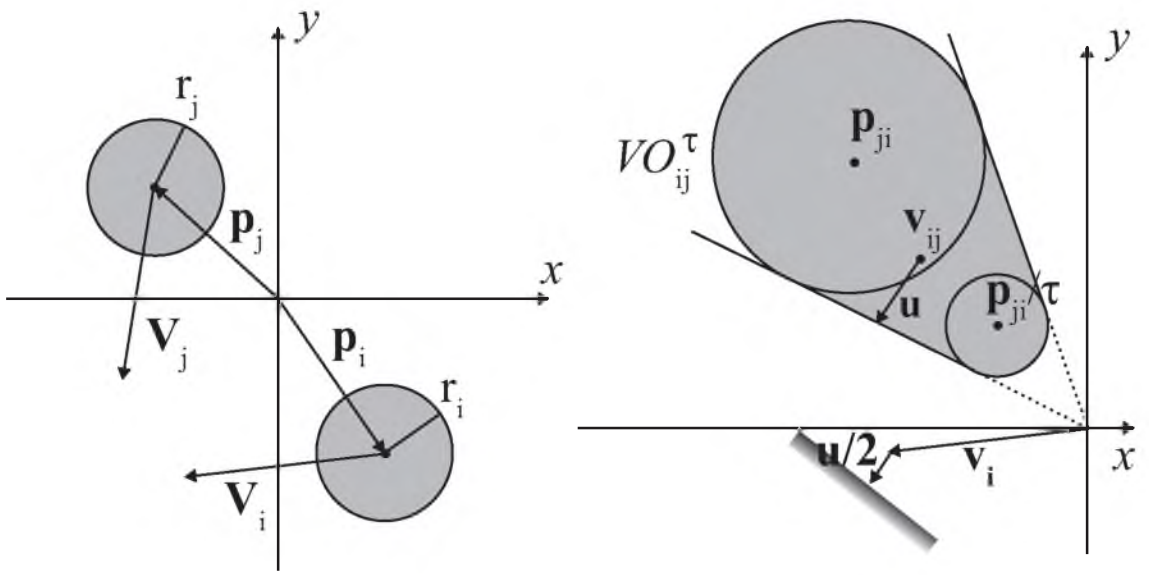


Figure 4.2. Shown on the left is an example configuration for two robots, i and j, which will lead to the velocity obstacle shown on the right. The updated velocity for robot i is shown to be updated by one-half of $\mathbf{u}$ as designed in ORCA.

4.1.2 Extension to 3-D

The development of *ORCA* was preformed in a 2-D environment. However, the ideas easily expand to a 3-D workspace. The bounding disc representative of a safe area for a robot can be replaced with a sphere or ellipsoid. The velocity obstacle in 2-D is viewed as a flat conic shape trending towards the origin and truncated by an arc due to the time horizon τ (obstacle moves toward origin for increasing time). In 3-D environments this obstacle becomes a traditional 3-D cone truncated by a spherical surface rather than an arc, such as in Fig. 4.3.

4.1.3 Noncooperative Agents and Exponential Convergence

One key assumption for *ORCA* to guarantee collision avoidance is that the agents sharing the workspace are cooperative and will perform the necessary reciprocal action to avoid collision. In practice, this assumption may be violated, for example, through loss of on-board tracking of another agent. The robustness of the algorithm allows for collision avoidance to still occur through exponential convergence to a collision free-path if only one robot is reacting.

We show this using an extreme case in which there are two robots, i and j, where robot i is using ORCA and robot j is a noncooperative agent moving along a constant velocity $\mathbf{v}_j$ with no regard to robot i. Robot i will create a velocity obstacle given a current configuration and, per *ORCA*, update its velocity to avoid collision. However, robot i is expecting the other agent to perform half of the necessary action. Therefore, in the first adaptation, robot i is not controlled completely out of collision, but instead only makes half the necessary change. Assuming an infinitesimally small time step, in the next sensing-action cycle, robot i creates (almost) the same velocity obstacle, but the necessary change in relative velocity between the robots is half of what it was in the previous cycle, and again robot i takes care of half of this necessary change. As a result, the velocity of i converges to a collision-avoiding velocity according to the difference equation

$$\begin{aligned}\mathbf{v}_i[t+\Delta t] &= \mathbf{v}_i[t] + \frac{1}{2}(\mathbf{v}^*_{ij} - \mathbf{v}_{ij}[t]) \\ &= \mathbf{v}_i[t] + \frac{1}{2}(\mathbf{v}^*_{ij} - \mathbf{v}_i[t] + \mathbf{v}_j),\end{aligned} \tag{4.2}$$

where $\mathbf{v}^*_{ij}$ is the relative velocity closest to the current relative velocity outside the velocity obstacle (see Fig. 4.4).

The difference equation solves to:

$$\mathbf{v}_i[t] = (\mathbf{v}^*_{ij} + \mathbf{v}_j) - 2^{-t/\Delta t}(\mathbf{v}^*_{ij} - \mathbf{v}_i[0] + \mathbf{v}_j). \tag{4.3}$$

Figure 4.3. Two robots, i and j, are shown to be on a straight-line collision course where each robot is moving with a velocity towards each other along the line intersecting their current positions. With such a velocity, a 3-D velocity obstacle can be constructed from the bounding ellipsoids as shown.

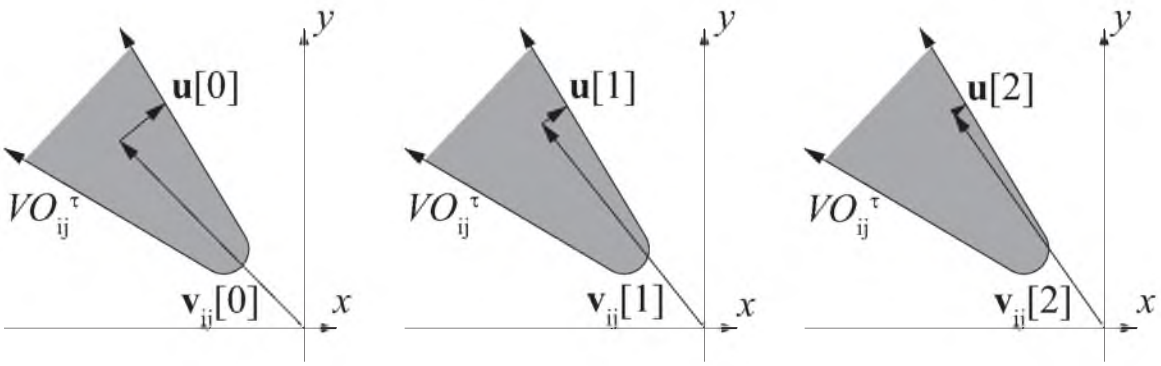


Figure 4.4. Assuming an infinitesimally small time-step, the velocity obstacle does not change from each previous time-step to the next. Only the relative velocity changes towards a free velocity each time-step. The velocity obstacle in three consecutive time steps is shown where $\mathbf{v}_{ij}$ converges exponentially towards a collision-free trajectory.

As $t \to \infty$, the velocity $\mathbf{v}_i$ of robot i converges to $(\mathbf{v}_{ij}^* + \mathbf{v}_j)$ and hence the relative velocity $\mathbf{v}_{ij}$ to $\mathbf{v}_{ij}^*$, which is a collision-avoiding relative velocity. As can be seen from Equation (4.3), this convergence happens with an exponential rate. A slight overestimation of the robot's radii is therefore enough in most practical cases to ensure collision avoidance even if not all robots act as assumed.

The same mechanism is at play when the robots have dynamics constraints that prevent them from acting as assumed. For instance, if the robots are not able to change their velocity instantaneously, but instead can only control their acceleration, we may let the acceleration be controlled as:

$$\mathbf{a}_i[t] = k_i \frac{1}{2}(\mathbf{v}_{ij}^* - \mathbf{v}_{ij}[t]), \tag{4.4}$$

for some control gain k_i. That is, the acceleration of robot i is set proportional to the required change in velocity for robot i as determined by ORCA. This leads to the differential equation:

$$\dot{\mathbf{v}}_i[t] = \mathbf{a}_i[t] = k_i \frac{1}{2}(\mathbf{v}_{ij}^* - \mathbf{v}_i[t] + \mathbf{v}_j), \tag{4.5}$$

for infinitesimally small time steps, which has as solution:

$$\mathbf{v}_i[t] = (\mathbf{v}_{ij}^* + \mathbf{v}_j) - e^{-k_i t/2}(\mathbf{v}_{ij}^* - \mathbf{v}_i[0] + \mathbf{v}_j). \tag{4.6}$$

Hence also in this case, the relative velocity $\mathbf{v}_{ij}$ converges to the collision-avoiding relative velocity $\mathbf{v}_{ij}^*$ with an exponential rate.

4.1.4 Sensing Uncertainty and Reciprocal Dances

A second key assumption for ORCA to guarantee collision avoidance is that there is perfect symmetry between robot i and j. More formally, the relative position $\mathbf{p}_{ji}$ and relative velocity $\mathbf{v}_{ij}$ as sensed by robot i are the exact negative of the relative position $\mathbf{p}_{ij}$ and relative velocity $\mathbf{v}_{ji}$ as sensed by robot j. In a perfect world, this is the case by definition, and as a result, we have that $\mathcal{VO}_{ij}^{\tau} = -\mathcal{VO}_{ji}^{\tau}$:

$$\mathbf{p}_{ji} = -\mathbf{p}_{ij},\ \mathbf{v}_{ij} = -\mathbf{v}_{ji} \ \rightarrow\ \mathcal{VO}_{ij}^{\tau} = -\mathcal{VO}_{ji}^{\tau}. \tag{4.7}$$

As can be seen from Fig. 4.2, when the relative velocity $\mathbf{v}_{ij}$ lies to the left of the center line of the velocity obstacle $\mathcal{VO}_{ij}^{\tau}$ (as seen from the origin), then the ORCA halfplane is constructed such that robot i will choose a velocity to pass robot j on its left. By symmetry, the relative velocity $\mathbf{v}_{ji}$ as seen by robot j lies to the left of the center line of $\mathcal{VO}_{ji}^{\tau}$ (again,

seen from the origin), and robot j will also choose to pass robot i on its left, leading to a smooth collision-avoiding motion by both robots.

More formally: Robot i selects a new velocity from its ORCA halfplane based on $\mathcal{VO}^{\tau}_{ij}$ and robot j selects a new velocity from its ORCA halfplane based on $\mathcal{VO}^{\tau}_{ji}$ such that:

$$\mathbf{v}_i[t+\Delta t] = \mathbf{v}_i[t] + \frac{1}{2}\mathbf{u}_i, \tag{4.8}$$

$$\mathbf{v}_j[t+\Delta t] = \mathbf{v}_j[t] + \frac{1}{2}\mathbf{u}_j, \tag{4.9}$$

which when expressed as relative velocities can be seen as

$$\mathbf{v}_{ij}[t+\Delta t] = \mathbf{v}_{ij}[t] + \frac{1}{2}\left(\mathbf{u}_i - \mathbf{u}_j\right), \tag{4.10}$$

$$\mathbf{u}_i = -\mathbf{u}_j, \text{ given } \mathcal{VO}_{ij} = -\mathcal{VO}_{ji}, \tag{4.11}$$

$$\mathbf{v}_{ij}[t+\Delta t] = \mathbf{v}_{ij}[t] + \mathbf{u}_i. \tag{4.12}$$

So, indeed, the new relative velocity $\mathbf{v}_{ij}[t+\Delta t]$ is a collision-avoiding velocity.

However, in practice, this symmetry is broken due to sensor noise on each of the robots. Let $\mathbf{v}_{ij}$ denote the true relative velocity of robot i and j, then the sensed relative velocities $\tilde{\mathbf{v}}_{ij}$ by robot i and $\tilde{\mathbf{v}}_{ji}$ by robot j can be assumed to be:

$$\tilde{\mathbf{v}}_{ij} = \mathbf{v}_{ij} + \mathbf{m}_i, \qquad \mathbf{m}_i \sim \mathcal{N}(\mathbf{0}, M_i), \tag{4.13}$$

$$\tilde{\mathbf{v}}_{ji} = -\mathbf{v}_{ij} + \mathbf{m}_j, \qquad \mathbf{m}_j \sim \mathcal{N}(\mathbf{0}, M_j), \tag{4.14}$$

where $\mathbf{m}_i$ and $\mathbf{m}_j$ are the sensing noise of robot i and j drawn from a Gaussian distribution with zero mean and variance M_i and M_j, respectively. Similarly, the relative position may be sensed with noise, but we focus on the sensing uncertainty of the relative velocity here.

The result is that if the true relative velocity $\mathbf{v}_{ij}$ lies very close to the center line of $\mathcal{VO}^{\tau}_{ij}$ (meaning that the robots are on a "head-on" collision-course), then the relative velocity $\tilde{\mathbf{v}}_{ij}$ sensed by robot i may be to the right of the center line of $\mathcal{VO}^{\tau}_{ij}$, while the relative velocity $\tilde{\mathbf{v}}_{ji}$ sensed by robot j may be to the left of the center line of $-\mathcal{VO}^{\tau}_{ij}$, or vice versa. This causes the robot i to want to pass robot j on its right, and robot j to want to pass robot i on its left, and the new velocities chosen by the robots are most likely anew on a collision course. This asymmetry may repeat a number of time-steps, giving rise to a *reciprocal dance*.

More formally, the lack of symmetry prevents the assumption that $\mathbf{u}_i = -\mathbf{u}_j$. When $\tilde{\mathbf{v}}_{ij}$ and $\tilde{\mathbf{v}}_{ji}$ lie on the same side of the center lines of $\mathcal{VO}^{\tau}_{ij}$ and $-\mathcal{VO}^{\tau}_{ij}$, respectively (as seen from the respective agent), then $\mathbf{u}_i \approx -\mathbf{u}_j$, and collision avoidance will smoothly occur given the discussion of the previous subsection. However, if $\tilde{\mathbf{v}}_{ij}$ and $\tilde{\mathbf{v}}_{ji}$ lie on opposite

sides, then $\mathbf{u}_i \not\approx -\mathbf{u}_j$, and a reciprocal dance will occur. More precisely, a reciprocal dance is expected to occur when

$$\mathbf{u}_i \cdot \mathbf{u}_j > 0, \tag{4.15}$$

and the robots will choose a collision-avoiding trajectory when

$$\mathbf{u}_i \cdot \mathbf{u}_j < 0. \tag{4.16}$$

For a deterministic system with perfect symmetry, we have $\mathbf{u}_i = -\mathbf{u}_j$, so it is always the case that $\mathbf{u}_i \cdot \mathbf{u}_j < 0$. A set of relative velocity obstacles for robots i and j can be seen in Fig. 4.5 where the sensing error in relative velocity will lead to a reciprocal dance.

It should be noted that the reciprocal dance is also a very human phenomenon. When two people find themselves walking down a narrow corridor on an apparent collision course, both people try to avoid collision by stepping to one side or the other. However, without communication, there are times when both people move the same absolute direction, beginning a cyclic back and forth movement until eventually one person makes a move opposite of their opposer, allowing both to move on without collision. This same behavior was observed during our robotic experiments and can be explained by the presence of sensing noise in the context of the ORCA formalism. We note that any approach introduced before to accommodate sensing uncertainty in reciprocal collision avoidance cannot prevent the occurrence of reciprocal dances without mutual coordination or communication between robots.

4.2 Experimental Implementation

We performed our experiments on Parrot AR.Drone 2.0 quadrotors both in an environment with motion capture equipment (to record "ground truth" data on the motion of the robots) and in an environment without. These quadrotor helicopters were remote-controlled by human operators (i.e. the human operators set their preferred velocity) and exhibit dynamics that are far from the idealized assumption that new velocities can be adopted instantaneously. Each quadrotor has an on-board forward facing camera to detect other quadrotors (each quadrotor carries a tag for identification) and a downward-facing camera to estimate its own absolute velocity.

4.2.1 System Overview

Our system is set up as schematically shown in Figure 4.6 for each quadrotor. Using a joystick connected to a desktop computer, the human operator indicates the preferred velocity for the quadrotor it is operating. This preferred velocity is taken as input by ORCA,

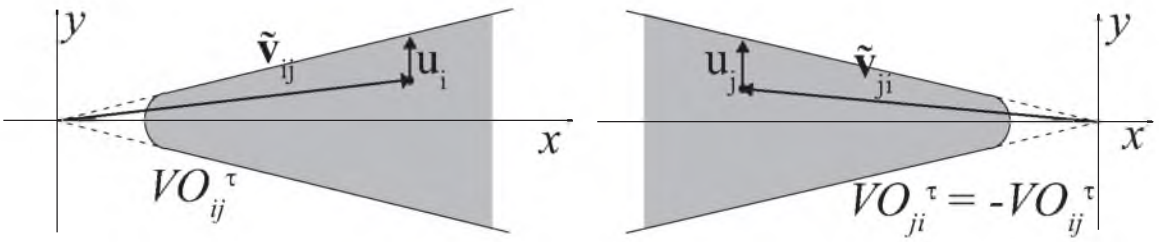


Figure 4.5. Given two robots, i and j, aligned along the global x-axis, a relative velocity obstacle can be created for each robot. The relative velocity obstacle is compared to the relative velocity to check for collision. Shown are two measured relative velocities which experience effects of uncertainty and are not symmetric. Given the asymmetry, if both relative velocities are on opposite sides of the velocity obstacle's centerline with respect to the origin, the robots will not avoid collision and undergo a reciprocal dance. Such an asymmetric measured relative velocity is shown.

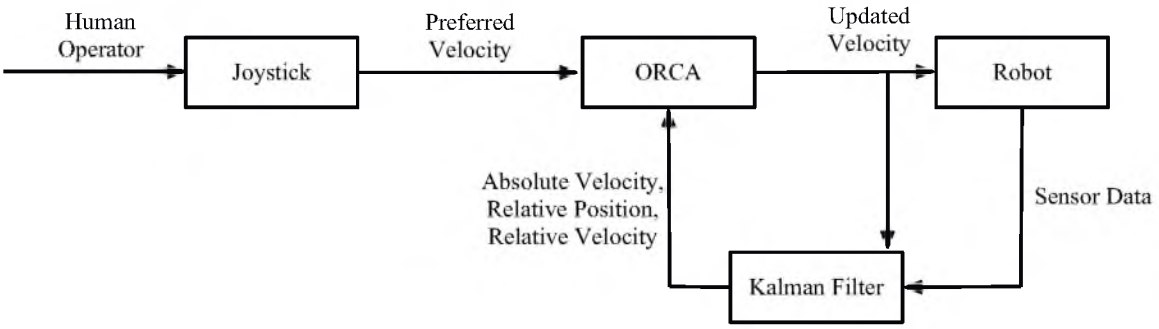


Figure 4.6. Shown is a schematic overview of our experimental system. A human operator controls each robot through a joystick, on a game controller for example. That user-desired velocity is checked in ORCA for collisions and if necessary, that desired velocity is updated to be a collision-free velocity. ORCA receives the relative position and relative velocity information, as well as individual absolute velocity, from the Kalman filter. The updated collision free velocity from ORCA is then sent to the robot's internal controller as well as to the Kalman filter. Sensor data from the actual robot are inputted to the Kalman filter as well.

which, assuming knowledge of the relative positions and relative velocities with respect to other quadrotors, outputs an actual safe velocity to adopt. ORCA runs on the desktop computer, as does the Kalman filter estimating the relative states of the two robots. This safe velocity is received wirelessly and processed with the AR.Drone's on-board computer (we use the standard software on-board the AR Drone 2.0 to abstract our discussion away from issues of quadrotor control), which steers the motors on the quadrotor to adopt this velocity. Note that due to quadrotor dynamics, this velocity is not assumed instantaneously. This input velocity, together with sensing data of the other quadrotors on the imaging plane of the forward-facing camera, is processed in a Kalman filter to estimate the relative positions and velocities of the other quadrotors. The absolute velocity of the quadrotor is estimated by the standard on-board software. This information is constantly fed to the ORCA module, which "transforms" the desired velocity of the operator to a velocity deemed safe by ORCA.

4.2.2 AR.Drone 2.0 - Parrot

We use the Parrot AR.Drone 2.0, a generally available off-the-shelf quadrotor platform for our experiments. Operation is generally conducted with a software development kit (SDK) released from the manufacturer [84]. The IMU runs at 200 Hz, where the ultrasonic sensors operate at 25 Hz. The on-board Kalman filter maintains an estimate of the state of the quadrotor including the attitude and velocity. This data is aggregated from the vision system and strapdown sensors. The AR.Drone stabilizes via an on-board PID controller [64], which acts on the orientation of the robot as a function of the on-board software's velocity estimate and the user input.

The digital signal processor runs a variety of proprietary and public computer vision algorithms, for example a Features from Accelerated Segment Test (FAST) corner detector. Its also provides pixel coordinates and a distance estimate to a special "tag" marker that can be attached to other objects as seen by the forward facing camera. Because the robot is a piece of propriety hardware, the specific methods and parameters concerning many of these video and stabilization algorithms is unpublished. The AR.Drone's hardware includes [64]:

- One GHz Cortex-A8 CPU with 256 MB RAM
- 800 MHz video digital signal processor
- 802.11n wireless link

- Nine degree-of-freedom IMU (composed of rate gyroscope, accelerometer, and magnetometer)
- Ultrasound and pressure sensors
- 30 Hz 720p forward-facing camera
- 60 Hz 320 x 240 pixel downward-facing camera

4.2.3 Kalman Filter for Relative Positioning

As discussed above, for reciprocal collision avoidance, each quadrotor i needs to know the relative position $\mathbf{p}_{ij}$ with respect to other quadrotors j, the relative velocity $\mathbf{v}_{ij}$ with respect to other quadrotors j, and its own absolute velocity $\mathbf{v}_i$. To keep track of these quantities, we implemented a Kalman filter for each quadrotor i with a simplified dynamics model:

$$\dot{\mathbf{p}}_{ij} = \mathbf{v}_i - \mathbf{v}_j, \qquad \forall (j \neq i) \tag{4.17}$$

$$\tilde{\mathbf{v}}_i = k(\mathbf{v}_i^* - \mathbf{v}_i), \tag{4.18}$$

$$\tilde{\mathbf{v}}_j = \mathbf{m}_j, \qquad \forall (j \neq i), \tag{4.19}$$

where $\mathbf{m}_j \sim \mathcal{N}(\mathbf{0}, M)$. Here, $\mathbf{v}_i^*$ is the new velocity for quadrotor i as output by the reciprocal collision avoidance algorithm; since we do not know this quantity for any other quadrotor, we assume the evolution of the other quadrotor's velocities to resemble a random walk with variance M.

Through the on-board camera, each quadrotor i measures the pixel coordinates $\tilde{\mathbf{b}}_j$ of other quadrotors j in its imaging plane, as well as their distance $\tilde{d}_j$ (using the size of the tag on quadrotor j). Also, a separate on-board Kalman filter that runs at a higher frequency keeps track of the full state of the quadrotor (velocity, orientation, angular velocity); this second Kalman filter provides "measurements" $\tilde{\mathbf{v}}_i$ of its velocity and $\tilde{R}_i$ of its orientation, that we will use to define the measurement model of the relative positioning Kalman filter. The measurement model is of the general form $\mathbf{z} = h(\mathbf{x}) + \mathbf{n}$, with $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, N)$, where $\mathbf{z}$ consists of $\tilde{\mathbf{v}}_i$ and $\tilde{\mathbf{b}}_j$ and $\tilde{d}_j$ for all $j \neq i$. The function h is defined by:

$$\tilde{\mathbf{b}}_j = \tilde{R}_i^T \mathbf{p}_{ij} f / (\tilde{R}_i^T \mathbf{p}_{ij})_z, \qquad \forall (j \neq i) \tag{4.20}$$

$$\tilde{d}_j = \|\mathbf{p}_{ij}\|, \qquad \forall (j \neq i) \tag{4.21}$$

$$\tilde{\mathbf{v}}_i = \mathbf{v}_i, \tag{4.22}$$

where f is the focal length of on-board camera (assuming it is a pinhole camera), and $(\mathbf{p})_z$ denotes the z-coordinate of vector $\mathbf{p}$.

4.2.4 ORCA

For the reciprocal collision avoidance in our system, we use the standard implementation of 3-D ORCA as publicly available on http://gamma.cs.unc.edu/RVO2/. Each quadrotor helicopter is geometrically modeled in ORCA as an ellipsoid encompassing the quadrotor elongated along the z-axis to discourage pairs of quadrotors to fly in each other's downwash from the propellers.

4.3 Results

In this section, we report results of our experiments. Firstly, we quantify the accuracy of the estimation of relative position and relative velocity using the on-board sensors and the Kalman filter by comparison to "ground truth" data obtained from a motion capture system. Second, we report on over one hundred runs with a pair of quadrotors that were controlled by human operators to fly into one another, and qualitatively describe the observed motion. None of our more than one hundred runs resulted in a collision between the robots.

4.3.1 Relative Position and Velocity Estimation

Human operators flew two quadrotors repeatedly on colliding trajectories. The motion capture system was used to collect a true position and velocity of each robot during the testing. The estimated values of position and velocity were collected from each robot. To quantify the accuracy of the Kalman Filter, the true relative position and estimated relative position were compared, shown in Fig. 4.7. The error associated with the estimated relative position is relatively large when an agent enters the viewing envelope at a far distance. As the agents approach each other, this error decreases in mean and standard deviation as shown. Given that the derivative of the relative position over time is used to estimate the relative velocity (see Eq. 4.17), the error in estimated relative velocity is directly coupled to the estimated relative velocity error.

As can be seen from the graph, the standard-deviation of the error increases with the distance between the quadrotors. This is expected, as the quadrotors are estimating distance with respect to each other based on the size (number of pixels) of the tag of the quadrotor in the imaging plane. Assuming a constant standard-deviation in the error of the pixel count, far-away quadrotors with a low pixel-count will have a relatively large standard deviation in the distance estimate. The graph also shows an underestimate of the true distance between quadrotors, with a bias that grows with the distance. Although not intentional, the on-board sensing provides conservative estimates of the relative position for the purpose of collision avoidance. Given the structured relation of the bias as a function of distance,

this bias can relatively easily be "calibrated out", but that was not deemed necessary for our purpose of collision avoidance.

4.3.2 Collision Avoidance Behavior

To validate the collision avoidance behavior, two quadrotors were flown at each other along a global x-axis at a height of approximately two meters off the ground. The experiment consisted of two users simultaneously controlling their robot straight forward on a collision-course with the other robot. The user continued to control the robot forward and allowed ORCA to change the motion to a collision-avoiding velocity. Over one hundred experiments of this type were completed. In these experiments, a reciprocal dance was observed in about 25% of the trials, a noncooperative agent was observed in 5% of the trials, and the remainder were smooth collision-avoiding trajectories. *None* of the experiments conducted resulted in a collision. A summary of these results can be seen in Table 4.1. Also, videos of some of the discussed experiments can be seen at `http://arl.cs.utah.edu/research/orca_quad/`.

Experiments were run in which the two quadrotors began within the sensing range of the forward-facing camera. Typically, these trials resulted in an expected collision-avoiding motion where each robot moved symmetrically about the vector between them, i.e. both agents move to left of the center-line as seen from their perspective. As the two quadrotors approach each other, ORCA begins to update the user-commanded velocity to avoid collision. ORCA avoids collision in this case by reducing the x-velocity slightly and inducing a y-velocity to move laterally. This action is shown in Fig. 4.8. As the quadrotors exit the collision region, the output from ORCA converged to the user-desired velocity from the human operator and the quadrotors continue to fly forward. An example of the expected trajectory was shown in Fig. 4.1.

At times during these experiments, the error in the relative position measurement manifested in reciprocal dances as previously hypothesized. In these cases, both quadrotors moved in the same true direction leading to a colliding trajectory. Typically, there was only one incorrect movement before both robots followed proper collision avoidance motions and moved symmetrically. These results were consistent with the hypothesis that sensing noise caused the reciprocal dances. As the robots move closer to each other, the sensing noise decreased and the robots were less likely to asymmetrically predict relative velocities. A sample reciprocal dance can be seen in Fig. 4.9.

In a small number of experiments, one of the quadrotors was unable to properly track the other quadrotor, leading to uncooperative behavior. Even in these experiments, no collisions were observed when a robot did not cooperate with the other. With no tracking

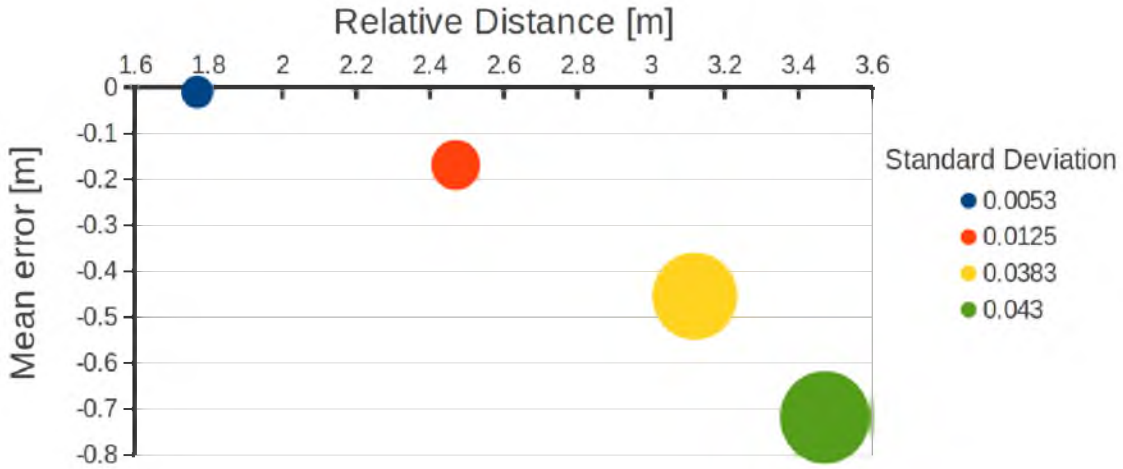


Figure 4.7. Quadrotors were flown in a straight-line trajectory by the user where ORCA was the sole method of avoiding collision. The graph plots error in the relative position measurement against the true relative position. As the robots come closer together, the sensing uncertainty exponentially converges to a small standard deviation with zero mean.

Table 4.1. Summary of Experimental Results

Behavior	# of Trials	% of Trials	# of Collisions
Reciprocal Dance	28	25.9%	0
Noncooperative	6	5.5%	0
Collision Avoidance	74	68.5%	0
Total	108	100 %	0

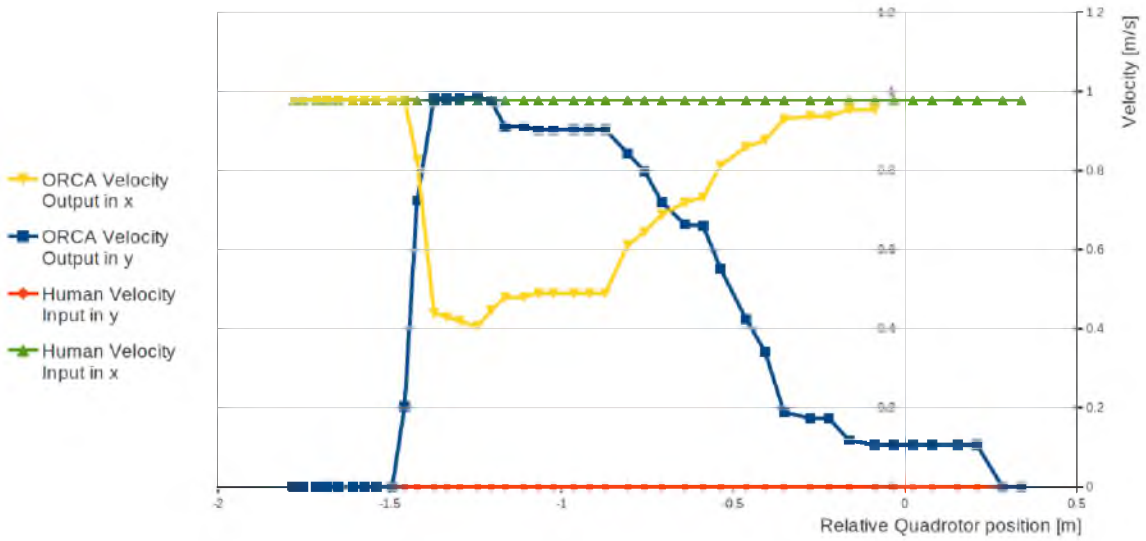


Figure 4.8. In this experiment, the quadrotors were flown on colliding trajectories by human operators along the x-axis. Shown is the user's command velocity for a single quadrotor in x and y and the command after the collision check in ORCA. The change in the x and y velocity is due to ORCA finding an optimal noncolliding path.

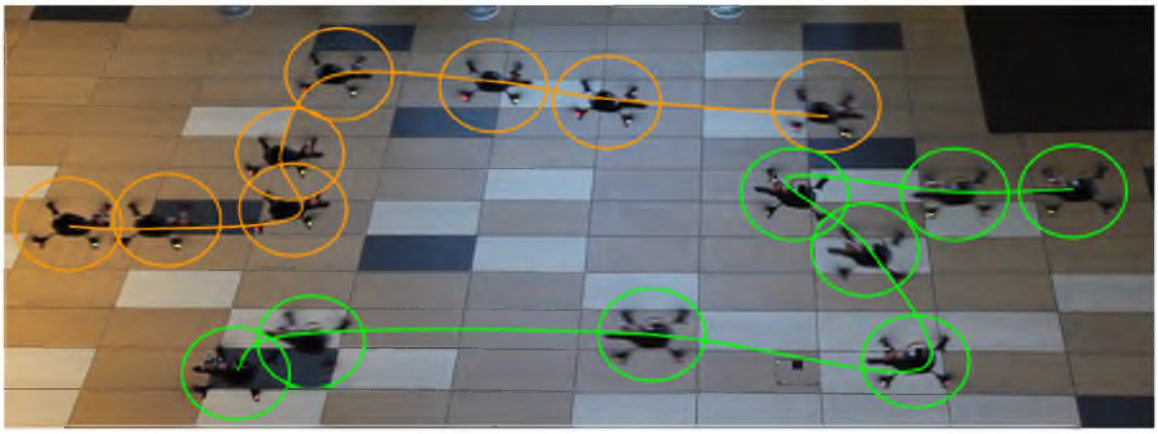


Figure 4.9. This experiment shows two quadrotors flown in a straight-line path by human operators. Due to sensing uncertainty, they undergo a reciprocal dance. Each robot initially flies in the same direction (up in the image) then, upon gaining a better relative velocity estimate, properly flies a noncolliding trajectory.

information being received from the quadrotors on-board software, the Kalman filter could not provide a relative position estimate and, therefore, ORCA did not construct a valid velocity obstacle. As a result, the uncooperative robot remained on a straight-line trajectory as per the user's input. The second quadrotor, which did have tracking, properly avoided collision, further supporting our hypothesis that ORCA is robust to noncooperative agents through exponential convergence.

Experiments were also run to validate ORCA with noncooperative agents where a quadrotor without tracking was held stationary and a user-controlled quadrotor with tracking was flown directly at it along a straight-line trajectory. Such an experiment is shown in Fig. 4.10. Concurrent with other noncooperative experiments, the robot who properly constructed the velocity obstacle took actions to avoid collision.

The dynamics of quadrotor helicopters were found to make ORCA less optimal, requiring it to rely on its ability to exponentially converge to collision free paths. A slightly larger bounding radius was the only requirement to allow for robust collision avoidance given exponential convergence to optimal collision avoidance. The fact that no collisions were observed in over one hundred experiments suggests that ORCA can provide reliable collision-avoidance even if the algorithm is not adapted specifically to account for the dynamics of the quadrotors or the sensing noise.

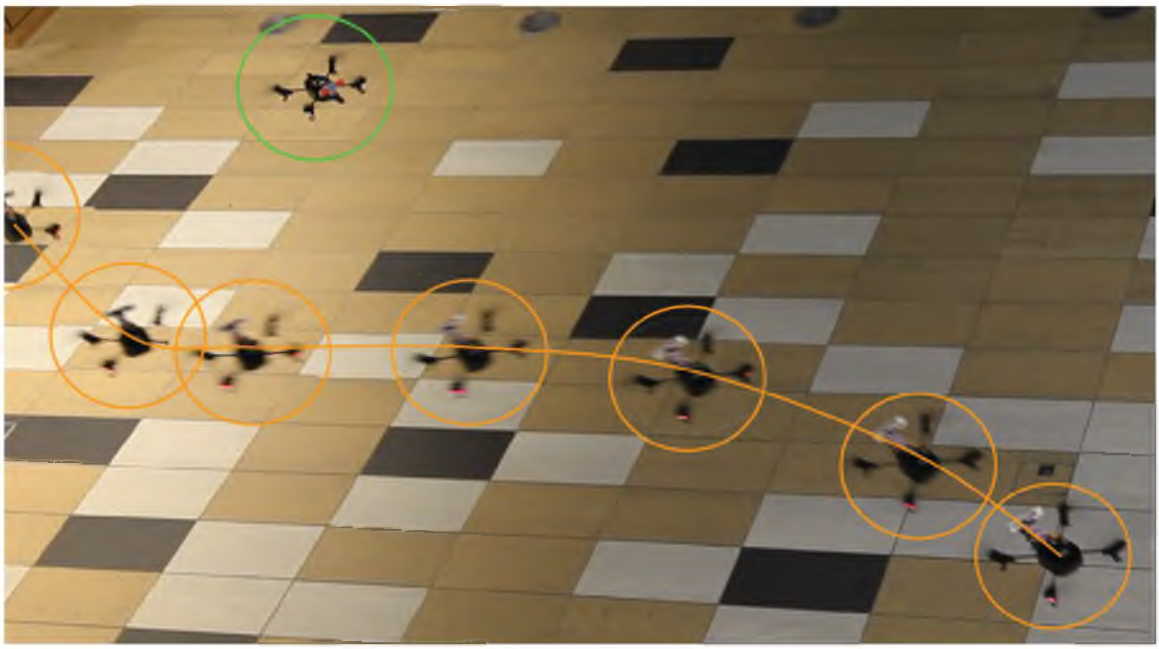


Figure 4.10. In this experiment, the top quadrotor, shown in orange, did not track the other quadrotor, shown in green. As expected, the bottom quadrotor flies a noncolliding velocity as a direct result of ORCA's ability to exponential converge to a collision-free trajectory given a noncooperating, colliding, robot.

CHAPTER 5

DESIGN CONSIDERATIONS OF THE ORCA EXPERIMENTS

5.1 Experimental Parameters

In the ORCA experiments, a variety of robot- and algorithm-specific parameters were selected to make the robot and experiment behave as desired within the ARL flight volume. The safety radius surrounding the quadrotor was experimentally found from a series of tests focused on limiting the aerodynamically interaction between the robots as they passed each other. The time horizon term τ that determines how far in the future collision will be considered was found according to the maximal thrust of the quadrotor. This maximal acceleration is a function of the maximal quadrotor angle relative to gravity that was allowed.

5.1.1 AR.Drone-specific Parameters

The AR.Drone features many settings that affected its use. These settings are configured from a batch of parameters sent to the robot when it is initialized.

The mass of the robot is a function of the configuration in which it was flown, and for our experiments, we decided to fly the robot without its hull. In lowering the mass of the robot, we were able to maximize its flight time. Also, we found that removing the indoor hull, due to its ducting geometry, increased the effective thrust from each of the motors. Each of these factors improved the efficiency of the robot, allowing for more experiments before the batteries needed to be changed.

Foremost to our experiment is the maximal angle allowed by the on-board controller which is related to the maximal thrust according to Equation 5.1. This simple equation can be used because the AR.Drone calibrates the thrust as to keep the robot the same height off the ground, independent of the pitch angle.

Where:

F is the total thrust of the aircraft

α is the "pitch"angle from the world $\hat{x}$ to the local $\hat{x}$ measured about the local $\hat{y}$
G is the nominal thrust required to bring the robot to a stationary hover

$$F = (sin(\alpha) * G) \tag{5.1}$$

With an AR.Drone mass of 400 grams, two maximal angles were used depending on the environment. In the ARL, a maximal angle of five degrees was used so that the acceleration of the aircraft was limited to 0.85 m/s^2. This limited the total force given by the robot to $\frac{1}{3}$ N. With a time horizon of two seconds and constant acceleration, the quadrotor could achieve velocities of 1.70 m/s from hover to avoid collision. The maximal deflection in this time would be 1.7 meters. This configuration allowed the AR.Drone to more than safely avoid collision given the 1.2 meter radius used to avoid aerodynamic interaction. Due to the conclusions of exponential convergence, higher angles would allow for safer flight, given the quadrotor could accelerate to a desired collision free velocity in a faster time frame.

In the larger environment, high speeds were desired, so the robot was allowed a maximal angle of 15 degrees. In these experiments, the maximal acceleration was 2.54 m/s^2, or the force of about 1 N. With a time horizon of two seconds and constant acceleration, this allows the quadrotor to deviate 5 meters from a hover position to avoid collision. The maximal speed within this distance would be 5 m/s.

5.1.2 Quadrotor Agent Radius

A safety radius was used in the experiments so that passing quadrotors did not affect each other. This was necessary because the model used did not take into account any aerodynamic interaction between the quadrotors. A feasible radius was determined from experiments preformed in the ARL. In these experiments, two AR.Drone quadrotors were brought to a hover state at the same height off the ground and then flown past each other at increasingly close distances until one or both became unstable. It was determined from these tests that a radius of about 1.25 meters was needed to avoid aerodynamic interaction. While this distance is quite large compared to the robot, the radius needed to account for the worst case configurations where a robot vectors thrust at the other robot in an attempt to quickly fly away from it. These experiments were preformed at the speeds, and thus thrust levels, seen in both environments of the ORCA experiments.

Tests were also preformed in 3-D where the quadrotors were actuated in various configurations at different heights. It was quickly seen that quadrotors flying over each other at distances less than about 3 meters created turbulent pressure gradients causing one or both

quadrotors to become unstable. The safety radius in 3-D was extended 2 meters above and below the quadrotor so they would not attempt to fly on top of the other. It is because of this effect that experiments in 2-D and 3-D often looked similar.

5.1.3 ORCA Time Horizon

The time horizon τ determines the time frame in which collision are avoided. The most important consideration that led to a time horizon of two seconds was the ability for the AR.Drone's front facing camera to resolve the incoming robot. From our analysis of the error profile of the camera sensor, we determined a maximum resolving distance of about 3.5 meters. The AR.Drone, given a chosen maximum velocity of 1.7 m/s, was desired to only consider collisions within this time frame. The time horizon constant of two seconds was rounded from the time it takes for the AR.Drone to travel at full speed through its maximum resolving distance. This is seen in the formation of the velocity obstacle where the position and radius of the leading edge of the obstacle is defined according to Equation 5.2.

Where:

$r_{leading}$ is the radius of the leading edge of the velocity obstacle

r_a is the radius representing agent A

r_b is the radius representing agent B

$p_{leading}$ is the position of the center of the radius of the leading edge of the velocity obstacle

p_a is the relative position of agent A, as measured from agent B

p_b is the relative position of agent B, as measured from agent A

τ is the time horizon

$$r_{leading} = (r_a + r_b)/\tau \tag{5.2}$$

$$p_{leading} = (p_b - P_a)/\tau \tag{5.3}$$

Figure 5.1 shows a graphical representation of the VO and its construction methods. As explained above, the shape of the VO is important not only because it defines if a collision will take place, but at what time in the future such a collision will be detected. A large τ translates to a larger time horizon, thus avoiding collision further in the future. Because natural quadrotor dynamics limited the instantaneous velocity that the robot could achieve, we designed in enough time for avoiding a collision in the worst-case scenario.

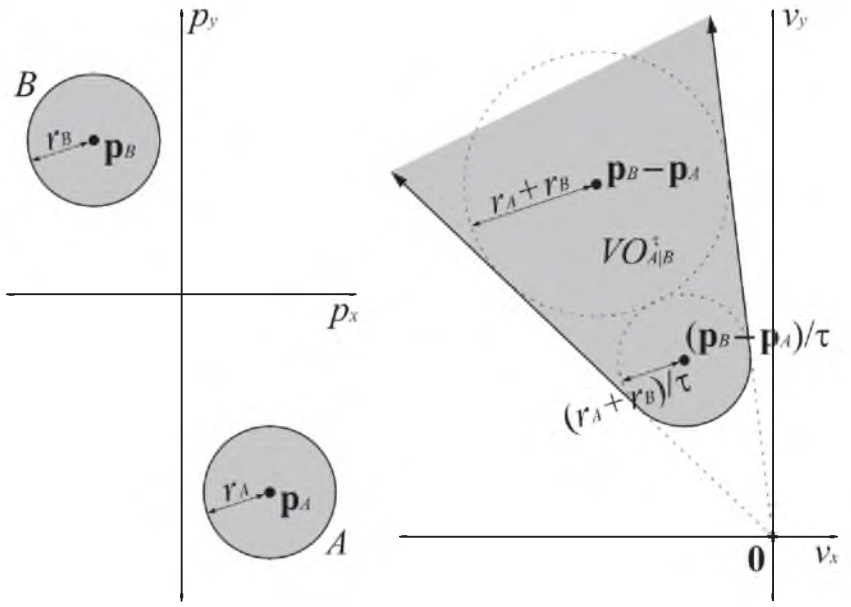


Figure 5.1. Graphical representation of the velocity obstacle.

CHAPTER 6

CONCLUSIONS

6.1 Conclusions from the ORCA Experiments

We have presented an analysis of Optimal Reciprocal Collision Avoidance on real robots using completely distributed acting and (on-board) sensing. With no centralized control or interrobot communication, two quadrotors were able to successfully avoid collision in experimentation using built-in CMOS cameras and tracking algorithms. We analyzed the effects of violating key assumptions of ORCA and how they affect our system. We showed that sensing uncertainty can lead to reciprocal dances. These reciprocal dances were seen during several of the experiment trials. In addition, our work exhibits that noncooperative agents can still avoid collision via exponential convergence. This was demonstrated at times when one of the two quadrotors did not properly construct the velocity obstacle, and was still able to avoid collision due to the other quadrotor's avoidance motion.

Our approach does have some drawbacks. The most notable is the use of the CMOS camera only allows tracking when the other robot is in a limited field of view. As the robots pass by each other and out of view, the Kalman filter uses the latest estimate and assumes this robot continues at that velocity. This assumption can obviously be violated due to the human control or external disturbances. Secondly, the estimated relative velocity from the Kalman filter is a derivative of the estimated relative position and as such can obtain very large values at times from noise in the position measurement. If this velocity estimates the robots flying quickly towards each other, it can cause the ORCA algorithm to behave as if the robots will collide imminently even when the robots are large distances apart. Often, this error is corrected in subsequent time steps, and the unnecessary motion will be small.

For the future, we would like to further investigate the sensing uncertainty which leads to reciprocal dances. A better understanding of this dance could lead to a reciprocal collision avoidance algorithm which is more robust to sensing uncertainty. An algorithm which could tolerate large sensing uncertainty would allow for cheaper robotic sensors to be implemented, helping advance the field of diminishing cost in robotics. Another possible avenue of research

is to investigate the improvement in collision avoidance capabilities from considering the dynamics of the system such as in [40].

6.2 Extensions of the ORCA Experiments

Given that the ORCA case study showed the robustness of the algorithm to violations of its native assumptions, many more tests can be constructed to begin extending ORCA to other avenues. While this list is not comprehensive, we hope that it can give a direction for new projects. Indeed the ARL was designed to test algorithms that are developed originally in simulation and the effect that real-world dynamics affect those algorithms. We hope that ORCA is only the first of many to be validated.

Using the free quadrotor yaw rotation, an agent could decide which other agents are most likely to be in colliding trajectories and decide to "look" at those colliding agents to take new measurements of their position. This research could work in swarms of large robots and focus on minimizing the uncertainty involved in sensing the velocity of colliding agents.

While some work has been done incorporating uncertainty into velocity obstacles, I believe that ORCA could be extended to deal with uncertainty. Even if the ORCA algorithm was left unchanged, research could be done into the effects of random and systematic error in the sensing of agents.

Work investigating the sensing fusion of the measurements into a concerted estimation of the robot state could lead to interesting versions of the Kalman filter. In much the same way the Kalman filter used in the experiment required switching between two distinct measurement models, we imagine the development of a method that would be able to take into account varying measurements regimes and sensors that operate at different rates.

6.3 Hindsight into the Construction of the ARL

It became apparent after completing the ORCA tests that, if given the option, the lab could be reconstructed to better serve its purpose. First of all, given what we know now, it would have been cheaper and more efficient to purchase the AR.Drone robots from the beginning. Although the lab was able to preform introductory experiments with the custom quadrotors, it became clear that it was not the effective choice given the laboratory's research goals. Buying an off-the-shelf robot makes much more sense when your focus is not the robot itself, but the actions it performs.

One of the major concerns with the laboratory is the need to constantly recalibrate the motion capture system. This is needed because minor changes in the configuration of the

cameras invalidates the calibration routines. The motion capture software assumes static cameras, and making the lab's hardware reflect that better would improve the system. I believe this could be accomplished one of two ways. Improvements to the truss structure could making it more rigid, and less prone to movements. This could be preformed by tensioning the truss to the ground via a system of metal wires set with lag bolts into the laboratory floor. The second method would involve setting motion capture cameras onto mounts into the walls, and hanging the safety netting from the ceiling. In this configuration, the cameras would connect to stands that go through the netting, but not touch it.

On the subject of motion capture, I also believe that the laboratory could use more cameras. Because of the limitations to the cameras in terms of maximal resolving distance between marker and camera, the cameras must be reconfigured depending on the desired workspace. Adding one more hub, and the associated four more cameras, to the group would allow the entire workspace to be utilized. One desires for each part of the volume to be resolved by three cameras so that a proper reconstruction can be preformed by the software. This takes into account the maximal resolving distance, and the 45 degree viewing angle of the V100:R2 cameras. Adding cameras would also speed up experiments, because the researchers would not need to preform the time consuming action of using the ladder to move the cameras.

The control of the experiments could also be improved with the construction of a control board. Such a board would interface with ROS and provide a series of controls such as buttons and analog sliders to control experimental parameters. Most critical to this controller would be the addition of an emergency stop button. Each experiment would feature a node listening for this stop command that would, for example, instruct the robots to instantly return to hover and then land.

Additional work could go into the lab to improve its use. A study into the optimal configuration of motion capture cameras could be completed to devise a method for efficient placement given a desired camera overlap and capture volume. Also, the use of real-time techniques could be implemented to improve robot control. Software systems such as real-time controllers can be used with ROS to limit latency, where additional dedicated hardware can improve the responsiveness of the any algorithm tested in the laboratory. These improvements will become necessary as more computationally expensive algorithms are investigated.

6.4 Conclusion

Special focus has been given to the construction of a software architecture that enables research to be completed quickly and efficiently. This modular structure allows for integration of packages written by other researchers due to the nodal format of ROS packages. This format will allow the addition of more robots through either manufacturer driver packages or custom nodes developed by ARL researchers. The first prototype uses a finite state machine where functional components were enabled with headers, where the second prototype uses standalone component packages. The various means in which each of the laboratory's robots are controlled requires that first architecture became increasingly complex as features needed to be added. The final architecture has been found to be much easier to use by those without a computer science background and an interested in robotics. The final and most important motivation for adopting the modular system was to shorten implementation time while allowing more flexibility in experimental styles and robot choice. Due to this, a variety of ground and aerial robots can be easily used in the lab.

The highlight of our physical experiments is the first hardware implementation of optimal reciprocal collision avoidance. We were able to show that this algorithm is a viable collision avoidance method in practice, since it was shown to be robust to violations of the algorithm's assumptions. Foremost, actual quadrotor dynamics deviate greatly from the infinite accelerations that the method assumes. The robots were observed to converge to collision free paths when put into colliding trajectories with robots not attempting to avoid collision. Natural reciprocal dances were seen in practice during experiments where state estimator error led to the optimal reciprocal collision avoidance algorithm instantaneously finding another colliding trajectory. These nonideal 'dance' paths are considered a worst-case scenario; however, the lack of collision supports the robustness of the method.

The results of this work are a viable mobile robotics research laboratory which has all of the capabilities of those found at larger robotics groups. Over these two years, the lab has developed quickly by focusing on the creation, implementation, and simulation of planning and sensing algorithms. As concepts move from invention to simulation and finally to physical systems, we expect the foundation developed in this thesis to be the backbone for transitioning the techniques developed in the Algorithmic Robotics Lab into real-world applications.

APPENDIX A

CODE SAMPLES

The code for the ORCA experiments can be found at: `https://github.com/parcon`

A.1 Global Launch File

```
<launch>
<include file="$(find arl_ROA)/launch/multibot.launch">
  <!-- all vars that included.launch requires must be set -->
  <arg name="QUAD_NAME" value="quad1" />
  <arg name="QUAD_IP" default="192.168.1.20" />
  <arg name="FIRST_QUAD" value="TRUE" />
</include>

<include file="$(find arl_ROA)/launch/multibot.launch">
  <!-- all vars that included.launch requires must be set -->
  <arg name="QUAD_NAME" value="quad2" />
  <arg name="QUAD_IP" value="192.168.1.10" />
  <arg name="FIRST_QUAD" value="FALSE" />
</include>

</launch>
```

A.2 Experimental Launch File

```
<launch>
<arg name="QUAD_NAME" default="quad1" />
<arg name="QUAD_IP" default="192.168.1.1" />
<arg name="FIRST_QUAD" default="TRUE" />
<arg name="ORCA_3D" default="TRUE" />
<arg name="JOYstick" default="TRUE" />

<node name="Drone_Drivers" pkg="ardrone_autonomy" type="ardrone_driver"
    output="screen" respawn="true" clear_params="true" args="-ip $(arg QUAD_IP)">

          <param name="outdoor" value="0" />
          <param name="max_bitrate" value="4000" />
          <param name="bitrate" value="4000" />
          <param name="navdata_demo" value="FALSE" /> <!-- May need to be false
              for tag detection -->
```

```
          <!-- <param name="navdata_options" value="NAVDATA_OPTION_FULL_MASK" />
              May need to be used for tag detection -->
           <param name="flight_without_shell" value="1" />
           <param name="altitude_max" value="5000" />
       <param name="altitude_min" value="100" />
       <param name="euler_angle_max" value="0.136" /> <!--limit angle to 8
           degrees -->
       <param name="control_vz_max" value="700" />
           <param name="control_yaw" value="1.75" />
           <param name="detect_type" value="CAD_TYPE_VISION" /> <!-- value 2 for
               front camera (enemy color), value 3 - no detection, and value 4 -
               vertical camera roundel detection values. CAD_TYPE_VISION is 2d
               horizontal tags (hopefully the colored ones on the drone) -->
           <param name="enemy_colors" value="2" /><!-- Detect green,yellow, blue
               shell (1,2,3) -->
           <param name= "enemy_without_shell" value="0" /> <!-- Detect
               indoor/outdoor shells -->
         <!--  <param name="default_groundstripe_colors"
              value="ARDRONE_DETECTION_COLOR_ARRACE_FINISH_LINE" /> -->
               <param name="detections_select_h" value="32" />
           <param name="detections_select_v" value="0" /><!-- Bottom Camera
               detection disabled, was detections_select_v_hsync 128 -->
           <param name="do_imu_caliberation" value="true" />
           <param name="tf_prefix" value="$(arg QUAD_NAME)" />
       <!-- Covariance Values (3x3 matrices reshaped to 1x9)-->
       <rosparam param="cov/imu_la">[0.1, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0,
           0.1]</rosparam>
       <rosparam param="cov/imu_av">[1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
           1.0]</rosparam>
       <rosparam param="cov/imu_or">[1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
           100000.0]</rosparam>
</node>

       <node pkg="arl_ROA" type="kalman" name="Kalman_Filter" respawn="true"/>

       <group if="$(arg ORCA_3D)">
              <node pkg="arl_ROA" type="ORCA_3D" name="ORCA_3D" respawn="true"/>
       </group>

       <group unless="$(arg ORCA_3D)">
              <node pkg="arl_ROA" type="ORCA_2D" name="ORCA_2D" respawn="true"/>
       </group>

       <node pkg="arl_ROA" type="track_tf" name="Tag_to_TF" respawn="true"/>
       <node pkg="arl_ROA" type="tag_tf" name="Kalman_Tag_TF" respawn="true"/>

       <group if="$(arg JOYstick)">
              <node pkg="arl_ROA" type="drone_control" name="Drone_Controller"
                  respawn="true"/>
              <group if="$(arg FIRST_QUAD)">
                     <node pkg="joy" type="joy_node" name="Xbox_Controller"
                         args="/dev/input/js0" respawn="true"/>
              </group>
```

```
                <group unless="$(arg FIRST_QUAD)">
                        <node pkg="joy" type="joy_node" name="Xbox_Controller"
                            args="/dev/input/js1" respawn="true"/>
                </group>
        </group>

        <group unless="$(arg JOYstick)">
                <node pkg="arl_ROA" type="fwd_test_control"
                    name="Drone_fwd_controller" respawn="true"/>

                <group if="$(arg FIRST_QUAD)">
                        <node pkg="joy" type="joy_node" name="Xbox_Controller"
                            args="/dev/input/js0" respawn="true"/>
                </group>

                <group unless="$(arg FIRST_QUAD)">
                        <node pkg="joy" type="joy_node" name="Xbox_Controller"
                            args="/dev/input/js1" respawn="true"/>
                </group>
        </group>

        <node pkg="rxtools" type="rxplot" name="Debug_Monitor_in_x"
            args="state_post_KF[3] state_post_KF[6]"/> -->
<!--
        <node pkg="rxtools" type="rxplot" output="screen" name="rxplot" args="-t
            'Debug Information'
        -l VaX,Vbx,Vay,Vby
        -b 100 /state_post_KF/data[3],/state_post_KF/data[6]
            /state_post_KF/data[4],/state_post_KF/data[7]" respawn="true" />
        -->
        <node pkg="rxtools" type="rxplot" output="screen" name="rxplot2" args="-t
            'Debug Information 2'
        -l joy_velx,cmd_vel_x,joy_vely,cmd_vel_y,Batt
        -b 100 /joy_vel/x,/cmd_vel/linear/x /joy_vel/y,/cmd_vel/linear/y
            /ardrone/navdata/batteryPercent" respawn="true" />

        <node pkg="image_view" type="image_view" respawn="false"
            name="Front_Camera_Viewer" output="screen">
            <param name="autosize" type="bool" value="TRUE" />
            <param name="window_name" type="str" value="'$(arg QUAD_NAME) Front
                Camera'" />
            <remap from="image" to="ardrone/front/image_raw" />
</node>
</launch>
```

APPENDIX B

FIRST PROTOTYPE EXPERIMENTS

B.1 Prototype One Experiment Graphs

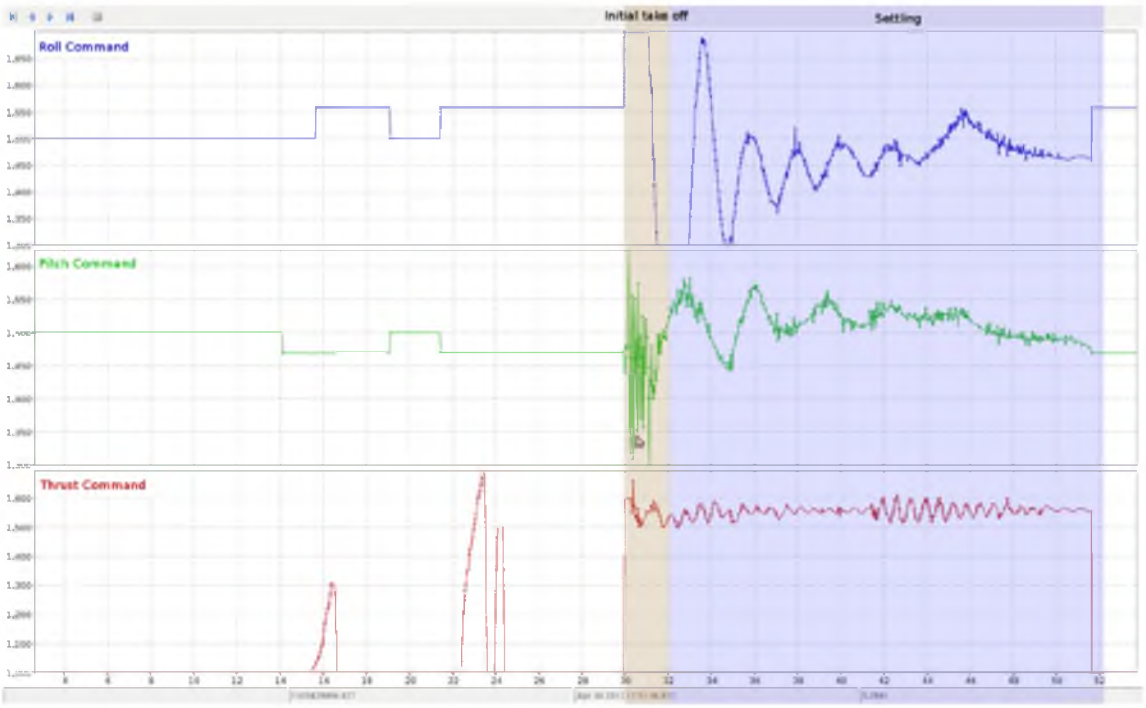


Figure B.1. Experiment One: X Step Command Signal.

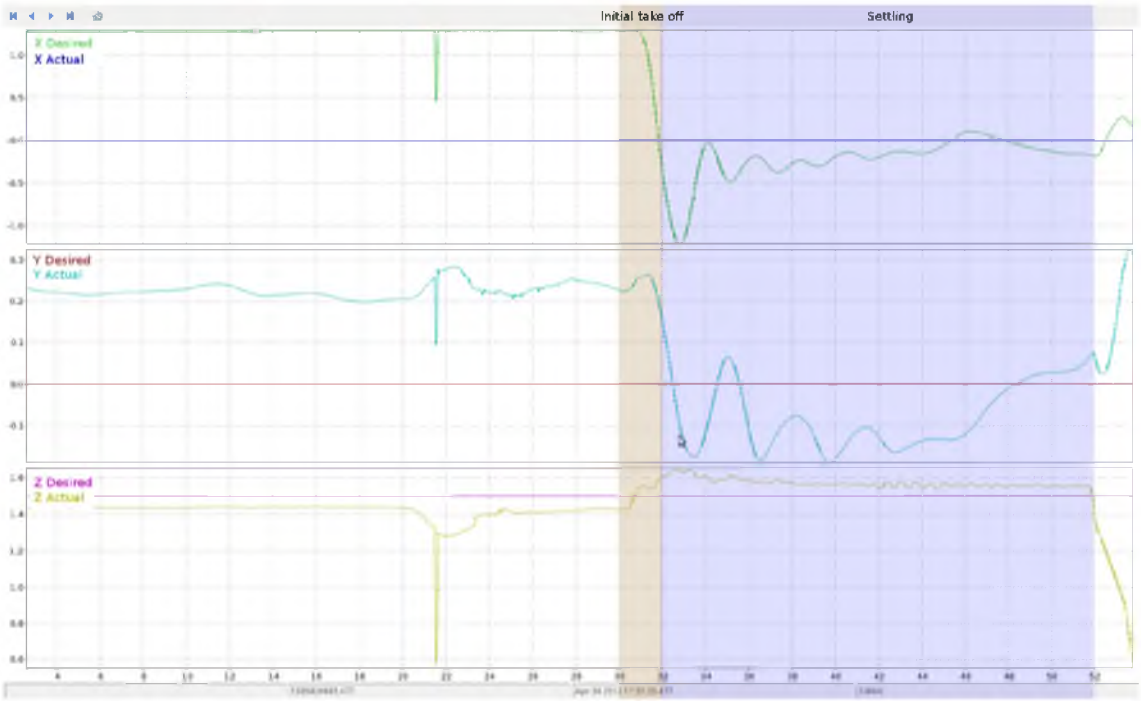


Figure B.2. Experiment One: X Step Position.

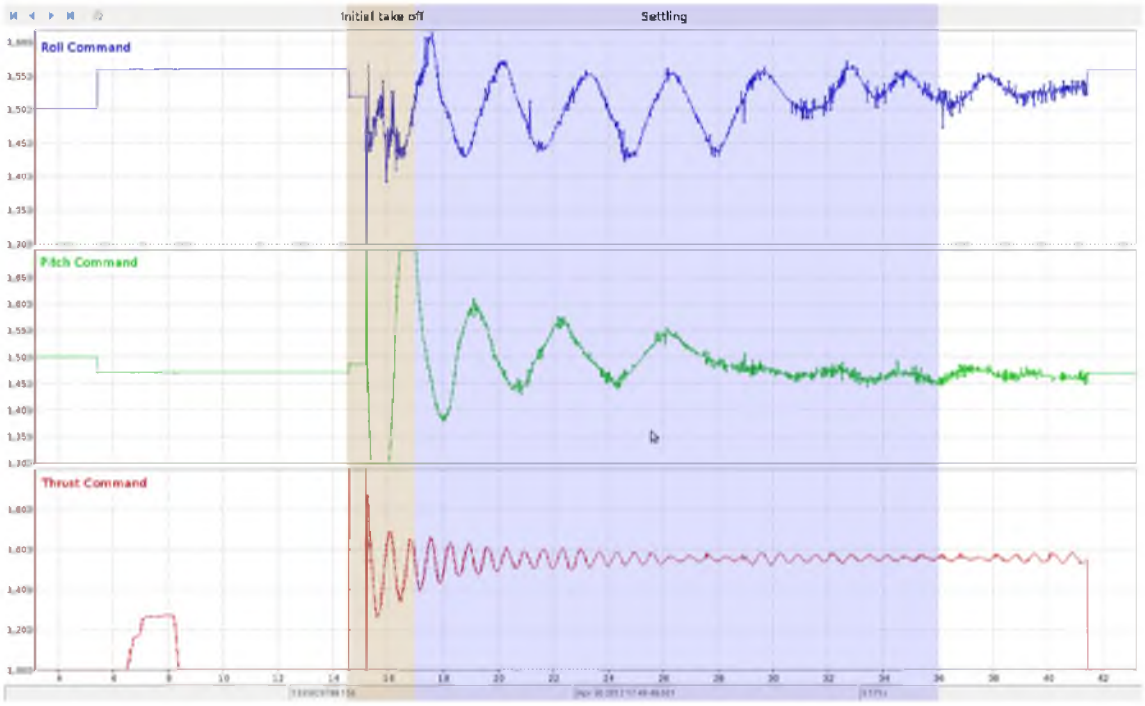


Figure B.3. Experiment Two: Z Step Command Signal.

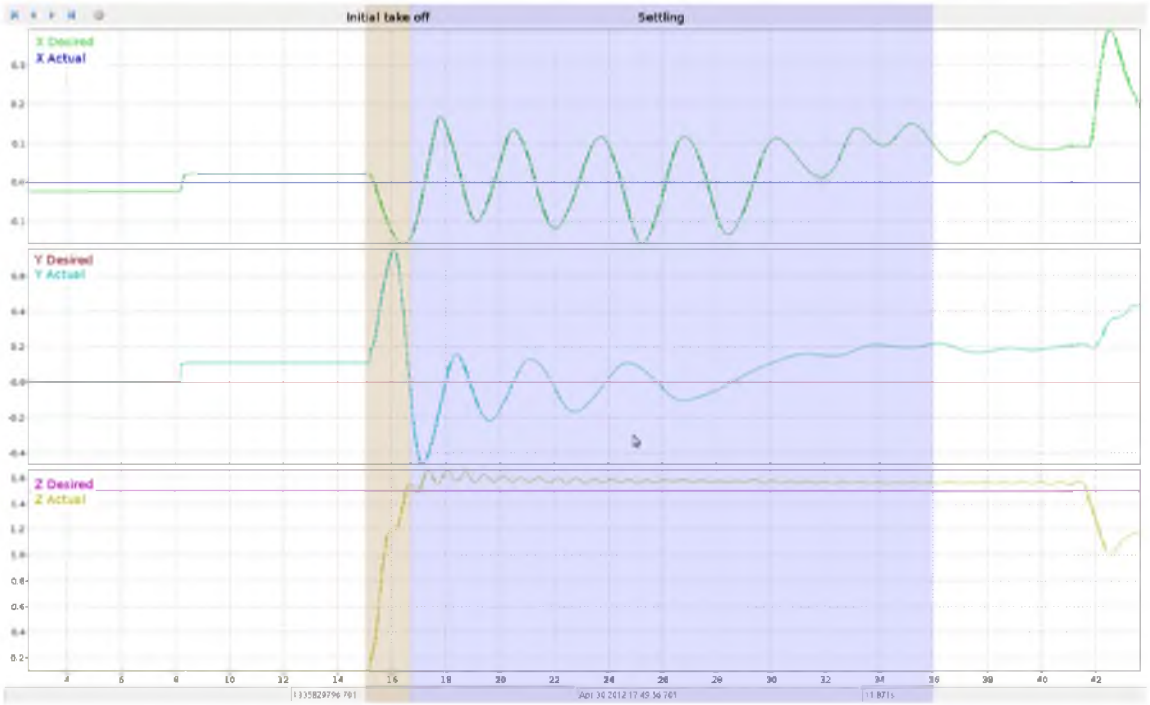


Figure B.4. Experiment Two: Z Step Position.

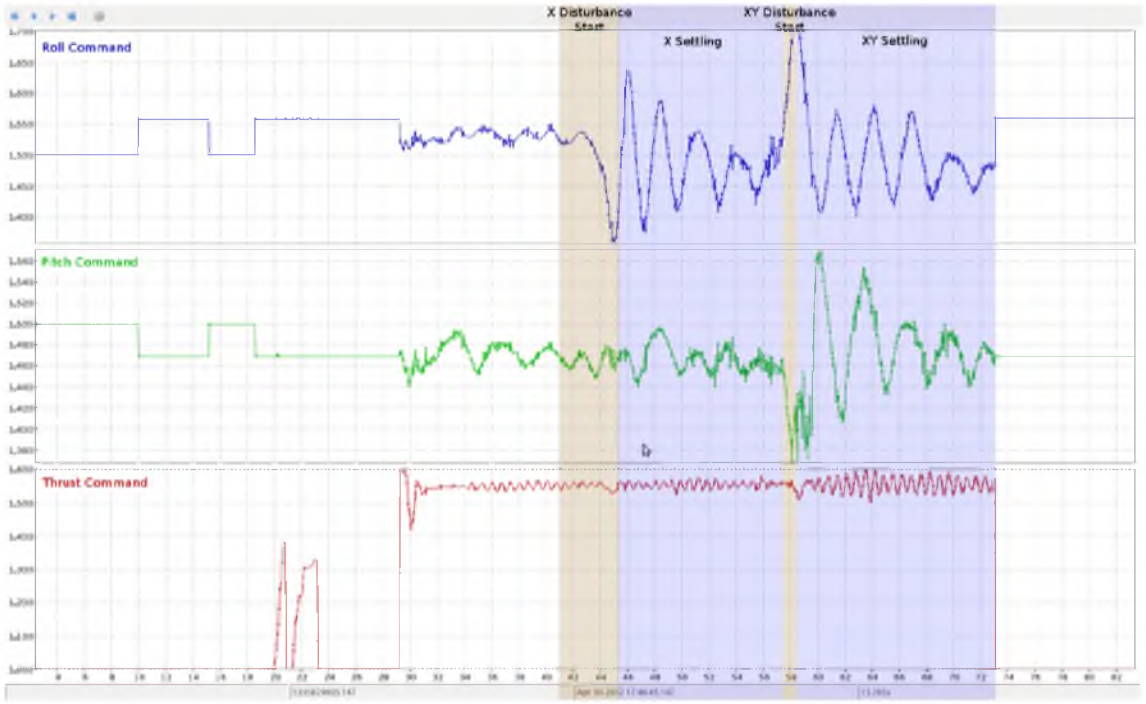


Figure B.5. Experiment Three: X and X-Y Disturbance Command Signal.

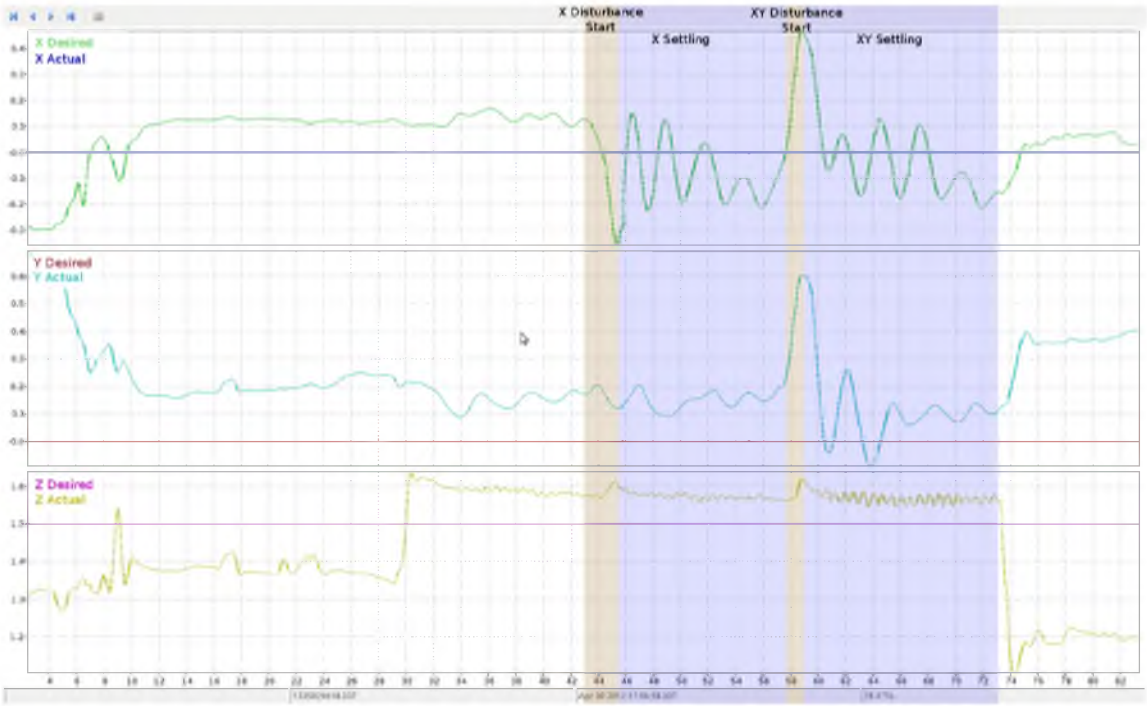


Figure B.6. Experiment Three: X and X-Y Disturbance Position

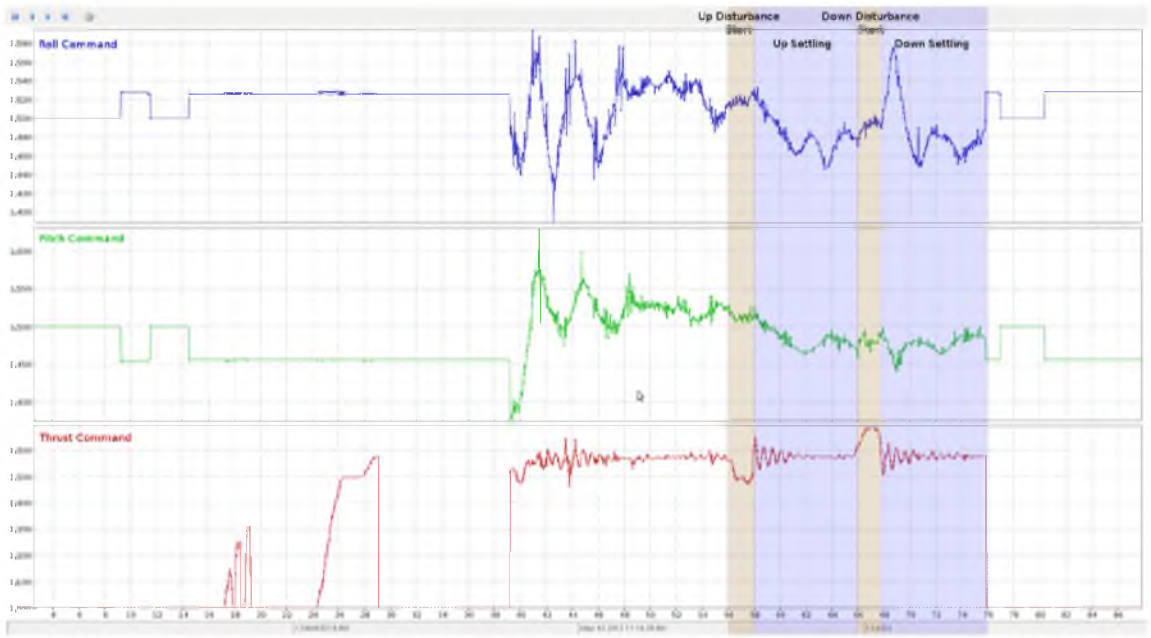


Figure B.7. Experiment Four: Z Disturbance Command Signal.

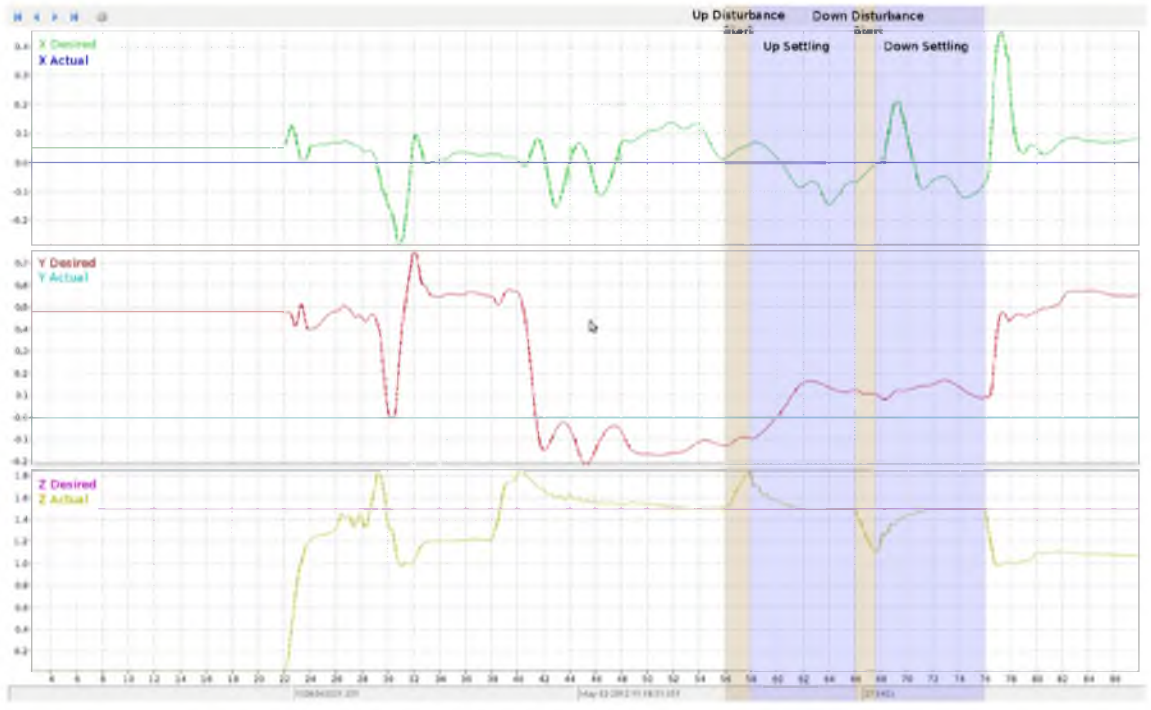


Figure B.8. Experiment Four: Z Disturbance Position

APPENDIX C

ARL PARTS LIST

C.1 ARL Parts List

Table C.1. Parts used in the Large Experimental Quadrotor

Type	Model
Frame	Gaui 330X
Battery	ZIPPY Flightmax 4000mAh 3S1P 20C LiPo
Propellers	Black 8 in. 2xCW and 2xCCW
Motors	ADH300L Brushless Outrunner 1100kv
AHRS	Ardupilot 2.0
Motor Controllers	FLYFUN 18A ESC

Table C.2. Parts used in the Small Experimental Quadrotor

Type	Model
Frame	Custom Polycarbonate
Battery	Turnigy nano-tech 1000mah 2S 25 50C Lipo
Propellers	Hobbyking 5030 Propellers (Red) - 2xCW and 2xCCW
Motors	18-11 2000kv Micro Brushless Outrunner
AHRS	Hobbyking Multi-rotor Control Board V3.0 (ATmega328 PA)
Motor Controllers	TURNIGY Plush 6A /.8bec

Table C.3. Support Materials

Type	Model	Number	Cost (USD)
Battery Charger	HobbyKing ECO6-10 200W 10A	2	40
Power Supply	HobbyKing 600W 17V Power Supply	1	67
Charge Bags	Lithium Polymer Charge Pack 18x22cm	4	2

REFERENCES

[1] H.-M. Huang, K. Pavek, B. Novak, J. Albus, and E. Messin, "A framework for autonomy levels for unmanned systems (alfus)," *Proceedings of the AUVSIs Unmanned Systems North America*, pp. 849–863, 2005.

[2] T. N. I. of Standards and Technology. (2012) Disclaimer: Use of nist information. [Online]. Available: http://www.nist.gov/public_affairs/disclaimer.cfm

[3] Orbotix. (2013) Copyright disclaimer. [Online]. Available: http://www.gosphero.com/company/fansite-guidelines/

[4] H.-M. Huang, K. Pavek, J. Albus, and E. Messina, "Autonomy levels for unmanned systems (alfus) framework: An update," in *Proceedings of the 2005 SPIE Defense and Security Symposium*, 2005, pp. 439–448.

[5] A. Steinfeld, T. Fong, D. Kaber, M. Lewis, J. Scholtz, A. Schultz, and M. Goodrich, "Common metrics for human-robot interaction," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM, 2006, pp. 33–40.

[6] S. Thrun, "Toward a framework for human-robot interaction," *Human–Computer Interaction*, vol. 19, no. 1-2, pp. 9–24, 2004.

[7] J. Gibson, "The concept of affordances," *Perceiving, acting, and knowing*, pp. 67–82, 1977.

[8] C. W. Nielsen, M. A. Goodrich, and R. W. Ricks, "Ecological interfaces for improving mobile robot teleoperation," *Robotics, IEEE Transactions on*, vol. 23, no. 5, pp. 927–941, 2007.

[9] K. Stubbs, P. J. Hinds, and D. Wettergreen, "Autonomy and common ground in human-robot interaction: A field study," *Intelligent Systems, IEEE*, vol. 22, no. 2, pp. 42–50, 2007.

[10] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-uav testbed," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 3, pp. 56–65, 2010.

[11] S. Lupashin, A. Schollig, M. Hehn, and R. D'Andrea, "The flying machine arena as of 2010," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 2970–2971.

[12] H. J. Kim and D. H. Shim, "A flight control system for aerial robots: algorithms and experiments," *Control engineering practice*, vol. 11, no. 12, pp. 1389–1400, 2003.

[13] D. A. Richards. (2013, May) Arthur's research blog: Flying robots. [Online]. Available: http://arthurrichards.blogspot.com/2013/04/flying-robots.html

[14] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 5. IEEE, 2004, pp. 4393–4398.

[15] S. Agostino, M. Mammone, M. Nelson, and T. Zhou. (2006) Classification of unmanned aerial vehicles. [Online]. Available: http://personal.mecheng.adelaide.edu.au/maziar.arjomandi/Aeronautical%20Engineering%20Projects/2006/group9.pdf

[16] S. D. Hanford, L. N. Long, and J. F. Horn, "A small semi-autonomous rotary-wing unmanned air vehicle (uav)," in *Proc. AIAA Infotech@ Aerospace Conf. Washington DC, USA*, 2005.

[17] H. Huang, G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 3277–3282.

[18] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," in *Proc. of the AIAA Guidance, Navigation, and Control Conference*, 2007, pp. 1–20.

[19] J. G. Leishman, *Principles of helicopter aerodynamics*. Cambridge University Press, 2006.

[20] P.-J. Bristeau, P. Martin, E. Salaün, N. Petit *et al.*, "The role of propeller aerodynamics in the model of a quadrotor uav," in *European Control Conference*, vol. 2009, 2009.

[21] J. E. Carryer, R. M. Ohline, and T. W. Kenny, *Introduction to mechatronic design*. Prentice Hall, 2011.

[22] P. Pillay and R. Krishnan, "Modeling, simulation, and analysis of permanent-magnet motor drives. ii. the brushless dc motor drive," *Industry Applications, IEEE Transactions on*, vol. 25, no. 2, pp. 274–279, 1989.

[23] K. Iizuka, H. Uzuhashi, M. Kano, T. Endo, and K. Mohri, "Microcomputer control for sensorless brushless motor," *Industry Applications, IEEE Transactions on*, no. 3, pp. 595–601, 1985.

[24] P. Martin and E. Salaun, "The true role of accelerometer feedback in quadrotor control," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1623–1629.

[25] J. van den Berg, "Introduction to quadrotor control," 2012.

[26] Willow Garage, "Robot operating system," 2010.

[27] A. Bargar, "A comprehensive ros interface for the aldebaran nao."

[28] D. Gossow, A. Leeper, D. Hershberger, and M. Ciocarlie, "Interactive markers: 3-d user interfaces for ros applications [ros topics]," *Robotics & Automation Magazine, IEEE*, vol. 18, no. 4, pp. 14–15, 2011.

[29] D. E. Chang, S. C. Shadden, J. E. Marsden, and R. Olfati-Saber, "Collision avoidance for multiple agent systems," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 1. IEEE, 2003, pp. 539–543.

[30] C. Cai, C. Yang, Q. Zhu, and Y. Liang, "Collision avoidance in multi-robot systems," 2007.

[31] N. A. Melchior and R. Simmons, "Particle rrt for path planning with uncertainty," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 1617–1624.

[32] K. I. Tsianos, I. A. Sucan, and L. E. Kavraki, "Sampling-based robot motion planning: Towards realistic applications," *Computer Science Review*, vol. 1, no. 1, pp. 2–11, 2007.

[33] J. van den Berg, M. Ling, and D. Mancha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *IEEE Int. Conf. on Robotics and Automation*, 2008.

[34] J. van den Berg, S. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Proc. Int. Symposium of Robotics Research*, 2009.

[35] J. Snape, J. van den Berg, S. Guy, and D. Manocha, "Smooth and collision-free navigation for multiple robots under differential-drive constraints," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, Oct., pp. 4584–4589.

[36] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," *Distributed Autonomous Robotic Systems*, pp. 203–216, 2010.

[37] J. Alonso-Mora, A. Breitenmoser, P. Beardsley, and R. Siegwart, "Reciprocal collision avoidance for multiple car-like robots," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 360–366.

[38] J. van den Berg, S. Guy, D. Manocha, and J. Snape, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *IEEE Int. Conf. on Robotics and Automation*, 2011.

[39] E. Lalish and K. A. Morgansen, "Distributed reactive collision avoidance," *Autonomous Robots*, vol. 32, no. 3, pp. 207–226, 2012.

[40] D. Bareiss and J. van den Berg, "Reciprocal collision avoidance for robots with linear dynamics using lqr-obstacles," in *IEEE Int. Conf. Robotics and Automation*, 2013.

[41] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using the relative velocity paradigm," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. IEEE, 1993, pp. 560–565.

[42] ——, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[43] J. Snape, J. van den Berg, S. Guy, and D. Manocha, "Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 5917–5922.

[44] ——, "The hybrid reciprocal velocity obstacle," *Robotics, IEEE Transactions on*, vol. 27, no. 4, pp. 696–706, 2011.

[45] Relic, http://www.spacemarine.com, THQ Inc.

[46] J. Snape, S. Guy, J. van den Berg, and D. Manocha, "Smooth coordination and navigation for multiple differential-drive robots," in *Proc. Int. Symp. on Experimental Robotics - ISER*, 2010.

[47] G. M. Hoffmann and C. J. Tomlin, "Decentralized cooperative collision avoidance for acceleration constrained vehicles," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on.* IEEE, 2008, pp. 4357–4363.

[48] T. Schouwenaars, J. How, and E. Feron, "Decentralized cooperative trajectory planning of multiple aircraft with hard safety guarantees," in *Proceedings of the AIAA guidance, navigation and control conference*, 2004.

[49] E. Lalish and K. A. Morgansen, "Decentralized reactive collision avoidance for multivehicle systems," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on.* IEEE, 2008, pp. 1218–1224.

[50] G. Roussos, G. Chaloulos, K. Kyriakopoulos, and J. Lygeros, "Control of multiple non-holonomic air vehicles under wind uncertainty using model predictive control and decentralized navigation functions," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, Dec., pp. 1225–1230.

[51] D. Hennes, D. Claes, W. Meeussen, and K. Tuyls, "Multi-robot collision avoidance with localization uncertainty," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1.* International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 147–154.

[52] D. Claes, D. Hennes, W. Meeussen, and K. Tuyls, "Calu: collision avoidance with localization uncertainty," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3.* International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 1495–1496.

[53] D. V. Dimarogonas and K. J. Kyriakopoulos, "Decentralized stabilization and collision avoidance of multiple air vehicles with limited sensing capabilities," in *American Control Conference, 2005. Proceedings of the 2005.* IEEE, 2005, pp. 4667–4672.

[54] S. Loizu, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Decentralized feedback stabilization of multiple nonholonomic agents," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 3. IEEE, 2004, pp. 3012–3017.

[55] I. NaturalPoint. (2013, May) Tracking tools software. [Online]. Available: http://www.naturalpoint.com/optitrack/products/tracking-tools/

[56] ——. (2013, May) Tracking tools software. [Online]. Available: http://www.naturalpoint.com/optitrack/products/tracking-tools/specs.html

[57] T. Niemueller, A. Ferrein, D. Beck, and G. Lakemeyer, "Design principles of the component-based robot software framework fawkes," in *Simulation, modeling, and programming for autonomous robots.* Springer, 2010, pp. 300–311.

[58] H. Bruyninckx, "Open robot control software: the orocos project," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3. IEEE, 2001, pp. 2523–2528.

[59] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.

[60] J. Jackson, "Microsoft robotics studio: A technical introduction," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 4, pp. 82–87, 2007.

[61] Z. Zhang, "Microsoft kinect sensor and its effect," *Multimedia, IEEE*, vol. 19, no. 2, pp. 4–10, 2012.

[62] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The claraty architecture for robotic autonomy," in *Aerospace Conference, 2001, IEEE Proceedings.*, vol. 1. IEEE, 2001, pp. 1–121.

[63] Wiki. (2013, May) Robots using ros. [Online]. Available: http://ros.org/wiki/Robots

[64] P.-J. Bristeau, F. Callou, D. Vissière, N. Petit *et al.*, "The navigation and control technology inside the ar. drone micro uav," in *World Congress*, vol. 18, no. 1, 2011, pp. 1477–1484.

[65] M. Ferguson, T. Foote, M. Wise, and D. Stonier. (2013, May) Robots: Turtlebot. [Online]. Available: http://ros.org/wiki/Robots/TurtleBot

[66] F. Tomik, S. Nudehi, L. L. Flynn, and R. Mukherjee, "Design, fabrication and control of spherobot: A spherical mobile robot," *Journal of Intelligent & Robotic Systems*, vol. 67, no. 2, pp. 117–131, 2012.

[67] W. Jingxia, "Application of zigbee/ieee 802.15. 4 protocol compatible rf module xbee/xbee pro [j]," *Electronic Engineer*, vol. 3, p. 008, 2007.

[68] P. Conroy, https://github.com/parcon, GitHub Respository.

[69] P. I. Corke *et al.*, *Visual Control of Robots: high-performance visual servoing*. Research Studies Press Taunton, Somerset, England, 1996.

[70] S. Bouabdallah, A. Noth, and R. Siegwart, "Pid vs lq control techniques applied to an indoor micro quadrotor," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2451–2456.

[71] A. Tayebi and S. McGilvray, "Attitude stabilization of a vtol quadrotor aircraft," *Control Systems Technology, IEEE Transactions on*, vol. 14, no. 3, pp. 562–571, 2006.

[72] G. Buskey, J. Roberts, P. Corke, P. Ridley, and G. Wyeth, "Sensing and control for a small-size helicopter," in *Experimental robotics VIII*. Springer, 2003, pp. 476–486.

[73] J. M. Roberts, P. I. Corke, and G. Buskey, "Low-cost flight control system for a small autonomous helicopter," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 1. IEEE, 2003, pp. 546–551.

[74] S. Colton and F. Mentor, "The balance filter," *Presentation, Massachusetts Institute of Technology*, 2007.

[75] A. M. D. Sheet, "publication 2549n-avr-05/11," *Atmel Corporation, San Jose, CA*, 2011.

[76] D. Group. (2013) Github repository for ardupilot. [Online]. Available: https://github.com/diydrones/ardupilot

[77] P. Martin and E. Salaün, "Design and implementation of a low-cost observer-based attitude and heading reference system," *Control Engineering Practice*, vol. 18, no. 7, pp. 712–722, 2010.

[78] W. Lum; Richard S.; (Redmond, WA) ; Guo; Wei; (Sammamish, "Device for the treatment of hiccups," Patent, 3 26, 2010. [Online]. Available: http://appft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=/netahtml/PTO/search-bool.html&r=1&f=G&l=50&co1=AND&d=PG01&s1=20100178984.PGNR.&OS=DN/20100178984RS=DN/20100178984

[79] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Incorporated, 2008.

[80] P. Conroy, D. Bareiss, M. Beall, and J. van den Berg, "3-d reciprocal collision avoidance on physical quadrotor helicopters with on-board sensing for relative positioning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems. Submitted.*, 2013.

[81] E. Goffman and P. Manning, *Relations in public: Microstudies of the public order*. Basic Books, New York, 1971.

[82] S. P. Hoogendoorn and W. Daamen, "Pedestrian behavior at bottlenecks," *Transportation Science*, vol. 39, no. 2, pp. 147–159, 2005.

[83] F. Feurtey, "Simulating the collision avoidance behavior of pedestrians," *Master's Thesis*, 2000.

[84] Parrot, "Ar.drone skd 2.0," https://devzone.parrot.com/wiki/oss-ardrone2/Listing.